# Entwicklung eines Audio-Controllers für ATEM Bildmischer

**Bachelor-Thesis**
zur Erlangung des akademischen Grades B.Sc.

**Eike Börgeling**
Mat.-Nr.: [redacted]

**Erstprüferin:**
Prof. Dr. Eva Wilk

**Zweitprüfer:**
Malte Sanders

Hamburg, 15. Juli 2022

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

## Abstract

In this bachelor thesis an audio controller for ATEM video mixers will be developed. The existing video mixers from the company *Blackmagic Design* will be investigated and analyzed. The work on the audio controller is partially based on a library for the developing board *Arduino* which will be extended during the work on this paper. Therefore methods of network analysis are used to investigate the communication protocol of the video mixers used in local network areas. Final tests will verify the correct implementation of the protocol in the library.

## Zusammenfassung

Die vorliegende Bachelorarbeit befasst sich mit der Entwicklung eines Audio-Controllers für ATEM-Bildmischer. Hierzu werden Bildmischer des Herstellers *Blackmagic Design* untersucht und analysiert. Die Arbeit basiert auf einer Bibliothek für die Entwicklungsplattform *Arduino*, welche im Verlauf erweitert wird. Dazu werden Methoden der Netzwerkanalyse angewandt um das Kommunikationsprotokoll der Bildmischer im lokalen Netzwerk zu untersuchen. Abschließende Tests verifizieren die korrekte Umsetzung des Protokolls in der Bibliothek.

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Codeverzeichnis

# 1. Einleitung

Seit einigen Jahren sind Livestream-Produktionen immer häufiger im Tagesgeschäft von Produktionsfirmen zu sehen. Viele Firmen entscheiden sich, Präsentationen oder Sitzungen online und vor virtuellem Publikum abzuhalten. Auch die ab 2020 grassierende Corona-Pandemie hat Auswirkungen auf die Videobranche: mehr und mehr Produktionen werden als hybride Veranstaltungen durchgeführt. Der Markt für Livestreaming wächst schnell [Futurebiz].

Im Jahr 2019 bringt der Hersteller *Blackmagic Design* den Bildmischer *ATEM Mini* auf den Markt. Verglichen mit den bisherigen Bildmischer-Modellen des Herstellers ist dieser mit einem Preis von 295 USD bei Erscheinen sehr günstig [BMD Media Archive]. Der Anspruch des kompakten Bildmischers ist, eine all-in-one Lösung zu bieten und schnelle Einsatzbereitschaft bei einfacher Bedienung durch Einsteiger und Amateure. Dies führt zu hohen Absatzzahlen und langen Wartezeiten. Spätestens jetzt halten Bildmischer in den darauffolgenden Jahren mehr und mehr Einzug in kleinere Produktionen und verpassen dem Thema Livestreaming einen Wachstumsschub.

Der *ATEM Mini* ist nun aus vielen Lagern von Produktionsfirmen nicht mehr wegzudenken. Durch den Einzug der Bildmischer in die Branche der Videoproduktionen hat sich inzwischen aber der Standard angepasst. Eine professionelle Arbeitsweise mit dedizierten Mitarbeitenden für Bildregie, Kamera und Ton ist oft gängige Praxis. Größere Bildmischer des Herstellers *Blackmagic Design* werden häufig schon bei kleinen Produktionen eingesetzt. Der *ATEM Mini* lässt sich nur bedingt in diese Arbeitsweise integrieren. Es fehlen ihm Funktionen und vor allem Bedienmöglichkeiten, die Audiosteuerung ist eine davon. Tontechniker:innen, welche den Komfort einen Mischpultes gewohnt sind, finden hier keine der bekannten intuitiven Steuerungsmöglichkeiten wieder.

Ziel dieser Bachelorarbeit ist es daher, einen Audio-Controller für ATEM Bildmischer zu entwickeln. Seine Aufgabe soll sein, die fehlenden Audio-Steuerungen auf dem *ATEM Mini* sinnvoll zu ergänzen. Dabei soll er sich nahtlos in die Produktionsumgebung integrieren und mit den Bildmischern per plug-and-play verbunden werden können. So sollen die Vorteile, wie geringen Abmessungen des *ATEM Mini* oder die einfache Bedienbarkeit für Einsteiger, durch den Audio-Controller ergänzt werden.

# 2. Einführung in die Thematik

Im folgenden Kapitel wird das Projektumfeld beleuchtet. Ein kurzer Überblick über die *ATEM Bildmischer* findet ebenso statt wie der Bericht über meine bisherigen Erfahrungen mit einem speziellen Modell, dem *ATEM Mini Pro*.
Des Weiteren zähle ich die benötigten Funktionen des Audio-Controllers auf, mit denen ein sinnvoller Betrieb mit den ATEM Bildmischern möglich ist.

## 2.1. ATEM Bildmischer

*Blackmagic Design* ist ein Hersteller von digitaler Videotechnik. Neben Kameras und Bildmischern gehören z.B. auch Speicherlösungen und Software zu ihrem umfangreichen Produktportfolio. Seit der Gründung 2002 ist das Unternehmen rasant gewachsen und zählt nun zu den führenden Herstellern in ihrem Segment [Blackmagic Unternehmen]. Einen großen Teil des Produktportfolios machen die Bildmischer aus. Sie reichen vom kleinen Modell für „Prosumer“[1] bis zum professionellen Equipment für Fernsehsender oder Ü-Wagen. Die Bildmischer tragen alle den Namen *ATEM* was möglicherweise vom Namen eines ägyptischen Gottes abgeleitet ist, sonst aber keine Bedeutung trägt [BMD Forum].


Quelle: blackmagicdesign.com - ATEM Mini, letzter Zugriff: 12.07.2022 https://www.blackmagicdesign.com/de/products/atemmini

Abbildung 2.1.: Die unterschiedlichen Ausführungen des ATEM Mini

Die für diese Bachelorarbeit relevanten Bildmischer sind die kleinsten Modelle *ATEM Mini*, *ATEM Mini Pro* sowie *ATEM Mini Extreme*, welche sich nur in wenigen Features voneinander

[1] englisches Kofferwort für einen Konsumenten mit professionellen Ansprüchen

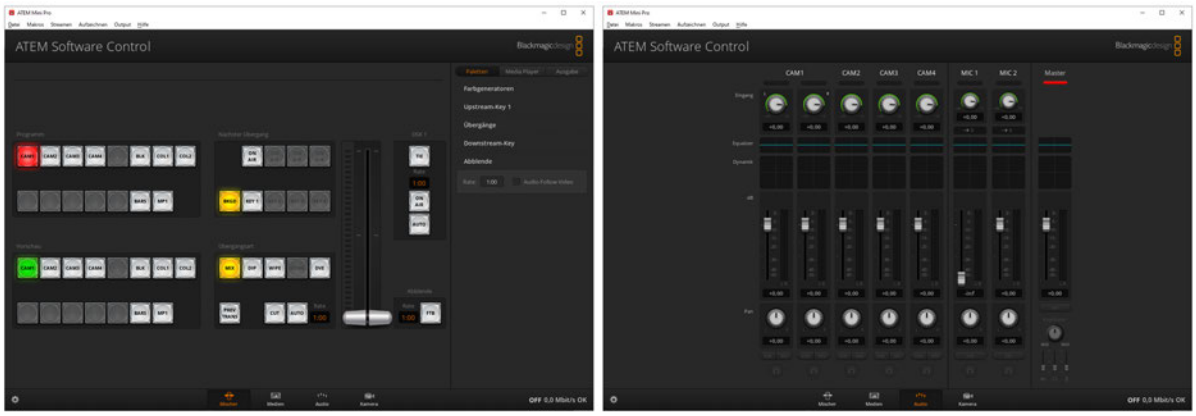
unterscheiden. Das Modell Mini Pro verfügt über vier HDMI-Kameraeingänge, einen HDMI Ausgang, zwei Stereo-Klinkeneingängen (3.5 mm) und USB- sowie Netzwerkanschluss (Ethernet). Es ist aufgrund seiner geringen Größe und Gewicht perfekt für mobile Einsätze geeignet. Die Bedienung findet direkt am Gerät statt, das heißt alle Kabel müssen zwangsläufig an den Platz der Bildregie geführt werden. Größere Bildmischer, wie die Produktserie *Constellation* oder *Production Studio* können vom Regieplatz räumlich getrennt werden, da die Bedienung nicht am Gerät selbst erfolgt. Die beiden Produktserien führen Modelle im 19 Zoll Format, welche sich nicht zur direkten Bedienung am Gerät selbst eignen. Stattdessen werden sie über ein Bedienteil gesteuert, welches mit dem Bildmischer über das lokale Netzwerk (Ethernet) verbunden ist (siehe Abbildung 2.2). Die Möglichkeit zur Steuerung der Bildmischer über das Netzwerk ist ein entscheidender Faktor in dieser Bachelorarbeit und wird im Kapitel 3.2 analysiert.

Quelle: blackmagicdesign.com - ATEM Advanced Panels, abgerufen: 04.07.2022
https://www.blackmagicdesign.com/de/products/atemconstellation/advancedpanel

Abbildung 2.2.: „Advanced Panel“ - ATEM Bedienteile in verschiedenen Größen

Auch über einen PC oder Mac lassen sich die ATEM Bildmischer umfangreich fernsteuern. Die dafür benötigte Software heißt *ATEM Software Control* und ist kostenfrei über die Hersteller-Website zu erhalten. Sie ist imstande verfügbare Mischer im Netzwerk zu finden und sie aufzulisten. Sofern nur ein Bildmischer gefunden wird, wird die Verbindung zwischen Software und Mischer automatisch aufgebaut. Das übersichtliche Interface der Software ist an die *Advanced Panel*, die Bedienteile der Rackmischer, angelehnt, passt sich aber dem Funktionsumfang der Verbundenden Mischer an. So sieht man bei aktiver Verbindung zum ATEM Mini lediglich vier Kameraeingänge auf der *Program-* und *Previewebene*, beim Steuern eines *ATEM Television Studio 4K* acht.

Neben den Videofunktionen lassen sich auch alle Audiofunktionen des Bildmischers steuern. Über einen weiteren Tab erreicht man den *Fairlight-Audiomixer* und erhält eine Übersicht über alle verfügbaren Kanäle und Einstellmöglichkeiten. Diese Ansicht ist Benutzer:innen der Software *DaVinci Resolve* bekannt, da dasselbe Interface zur Audiosteuerung verwendet wird. Der Fairlight Audiomixer ist in Kapitel 3.1 näher beschrieben.




Quelle: Eigene Darstellung

Abbildung 2.3.: ATEM Software Control: links Mischer Tab, rechts Fairlight Mixer

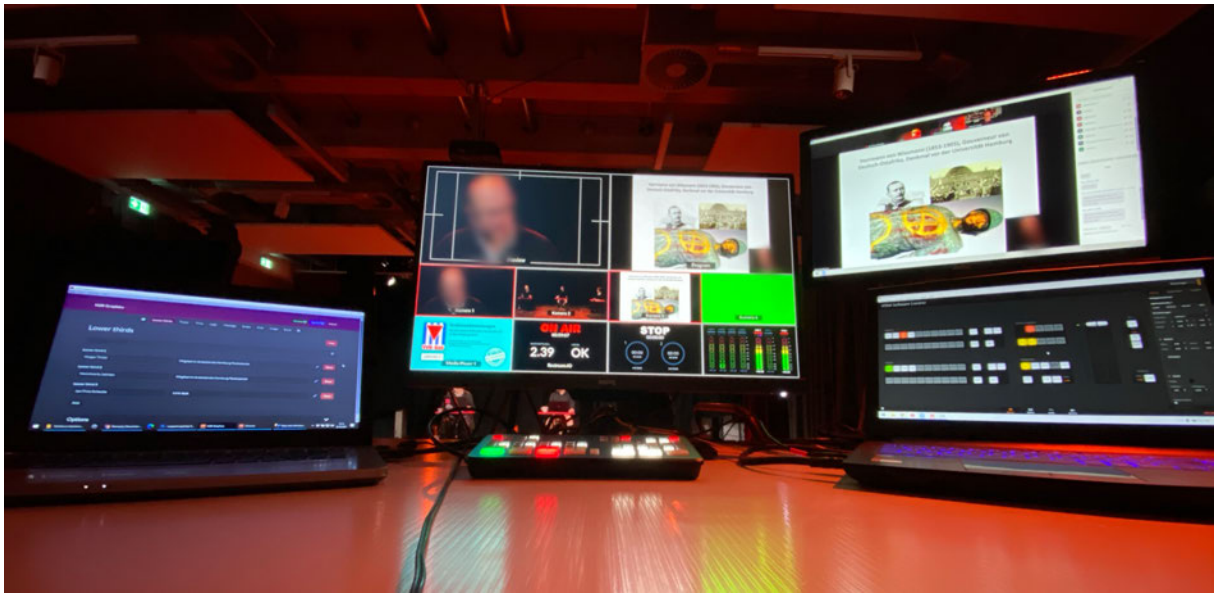
### 2.1.1. Praxiserfahrungen mit dem ATEM Mini Pro

Während mehrerer Live-Streaming-Produktionen konnte ich den *ATEM Mini Pro* einsetzen und bedienen. Ich stellte hierbei fest, dass er für diverse Arten von Produktionen gut geeignet ist. Sowohl für Präsentationen, welche den Anschluss von Powerpoint-Präsentationen an den Bildmischer erfordern, als auch für Interviews beziehungsweise Dialoge bei denen zwei Kameras oder mehr zum Einsatz kommen, ist der kleine Bildmischer gut geeignet. Seine vier Eingänge verfügen jeweils über Scaler, welche das Eingangssignal in die im Bildmischer eingestellte Videonorm wandeln und so eine sehr flexible Verschaltung ermöglichen. In einer Situation konnte ich einen Laptop, mit einem Ausgabeformat von `1920x1080p60` zusammen mit einer Kamera anschließen, welche ein Bild von `1920x1080i50 ausgab`. Beide Normen wurden vom Bildmischer in die eingestellte Videonorm von `1920x1080p25` gewandelt.

Auch während eines Livestream-Konzertes habe ich den Bildmischer eingesetzt. Die die Nähe der Kameras zur Bühne, und die Platzierung des Bildmischers beim FOH-Platz[2], mussten 20 m HDMI-Kabel eingesetzt werden. Trotz dieser Überschreitung der maximalen Länge von HDMI-Kabeln, konnte das Signal einwandfrei erkannt werden. Weiterhin wurde bei dieser Veranstaltung der Ton per Klinkeneingang in den Bildmischer geführt. Mithilfe des eingebauten Audio-Delays konnte die Tonspur um wenige Frames verzögert werden, was die Synchronizität zwischen Bild und Ton herstellte.

Der Vorteil der geringen Abmessungen von 230 mm Breite und 103 mm Tiefe ist gleichzeitig auch ein Nachteil. Durch das kleine Interface sind die Taster auf dem Bildmischer klein. Lediglich die wichtigsten Funktionen finden Platz auf der Oberfläche, was einen Einsatz der ATEM Software Control fast immer obligatorisch macht. So wird der Ton in jedem Kanal über einen Taster ein- und über einen weiteren Taster ausgeschaltet. Die Lautstärkeregelung lässt nur ein Steuern des Kanalfaders zu, welcher allerdings ebenfalls über Taster bedient wird. Zum Erhöhen der Lautstärke gibt es einen Taster mit Pfeil nach oben, zum Verringern einen Taster mit Pfeil nach unten. Diese Art der Lautstärkeregelung erinnert eher an ein Smartphone oder einen *MP3*-

[2]Der FOH-Platz, an dem die Ton- und Lichttechniker:innen stehen, ist üblicherweise gegenüber der Bühne, daher hat sich die Bezeichnung „front of house“ etabliert




Quelle: Eigene Darstellung

Abbildung 2.4.: Livestream einer Lesung mit dem ATEM Mini Pro und ATEM Software Control

*Player*, jedoch nicht an die Bedienung eines Mischpultes. Ein Tastendruck senkt bzw. erhöht die Lautstärke dabei um 3 dB, das Gedrückthalten faded den Kanal innerhalb von 17 s auf −100 dB.

Diese Steuerelemente machen eine Bedienung sehr schwierig, vor allem aber wenig intuitiv. Schnelles Reagieren mit dem Fader ist nicht möglich, das Steuern der Eingangsverstärkung ebenso nicht. Lediglich das live Schalten der Kanäle sowie die optische Rückmeldung darüber in Form eines rot beleuchteten Buttons funktioniert zuverlässig und intuitiv. Diese Umstände gaben die Idee zur Entwicklung des Audio-Controllers für die ATEM Bildmischer.

## 2.2. Benötigte Funktionen des Controllers

Zur intuitiven Steuerung des Bildmischers sind einige Funktionen des Controllers obligatorisch. Die folgende Tabelle 2.1 zeigt die geforderten Funktionen des Controllers sortiert nach Priorität. Die wichtigsten Funktionen sind hier mit Priorität „hoch“ versehen, diese sollten unbedingt umgesetzt werden.

Für das Mischen von z.B. zwei Mikrofonen bei einem Dialog, ist es notwendig, die Lautstärke mittels eines Faders einstellen zu können. Dies ist intuitiv und die Bedienenden sind einen Fader von Audio-Mischpulten in der Regel gewohnt. Die Eingangsverstärkung (oder Gain) ist eine ebenfalls wichtige Einstellung, welche auf dem Controller vorzunehmen ist. Durch die Möglichkeit diverse Mikrofone oder Quellen mit line-Pegel sowohl an Kameraeingänge (über HDMI) als auch an die Mikrofoneingänge anschließen zu können, müssen die Bedienenden in der Lage sein, schnell auf unterschiedliche Pegel reagieren zu können. So können die Pegel der Eingänge aufeinander angeglichen werden und ggf. eine Übersteuerung des Eingangs vermieden werden.
Bei den ATEM Bildmischern wird das Stummschalten (muten) eines Kanals anders signalisiert als bei Audio-Mischpulten. Oft nutzen letztere ein rotes Licht um einen stummgeschalteten Ka-

nal zu identifizieren. Der Bildmischer hingegen zeigt mit dem roten Licht an, dass ein Kanal live ist, was hier das Gegenteil von mute ist. Es besteht also Verwechslungsgefahr. Um eine einheitliche Linie zu finden wähle ich im weiteren Verlauf die Variante des Bildmischers, es kann ein Kanal also live geschaltet werden, was mit einem roten Licht signalisiert werden soll. Wenn dieser Kanal nicht live ist, zeigt er auch kein rotes Licht. Man spricht bei diesem Prinzip auch von einem Audio-Tally.

| **Funktion** | **Beschreibung** | **Priorität** |
|---|---|---|
| Fader-Lautstärke | Steuerung der Lautstärke eines Kanals | hoch |
| Gain | Einstellen der Eingangsverstärkung eines Kanals | hoch |
| Live/Mute | Der Kanal kann live bzw. mute geschaltet werden | hoch |
| Audio-Delay | Verzögerung des Kanals in Frames einstellen | mittel |
| Kanal Settings | Konfigurieren von Mono oder Stereo Quelle | niedrig |
| Balance | Kanal-Panning einstellen | niedrig |

Tabelle 2.1.: Benötigte Funktionen des Controllers sortiert nach Priorität

Die mit Priorität „mittel“ oder „niedrig“ aufgelisteten Funktionen sollen bei Bedarf umgesetzt werden. Für einen sinnvollen Betrieb sind sie keine Voraussetzung, würden jedoch den Controller vervollständigen und die Bedienung komfortabler machen. Alle Einstellungen, welche der Controller auf eine physische Ebene bringt, können weiterhin auch mit der Control-Software gesteuert werden.

## 2.3. Arduino

Ein wichtiger Teil dieser Bachelorarbeit ist die *Arduino Plattform*. Sie bietet open-source Hardware sowie Software in Form von Mikrocontroller-Boards und IDE (Integrated Development Environment). Die Boards basieren jeweils auf einem Mikrocontroller, sind aber zusätzlich mit erweiternden Hardwarefeatures ausgestattet. So kann ein *Arduino Uno* mit einem *ATMEGA328P* Mikrocontroller direkt über eine USB Schnittstelle programmiert werden, anstatt aufwändig in einen externen Programmer[3] gesteckt zu werden. Auch verfügen die Boards über periphere Elektronik zur Spannungsversorgung.

Neben der Hardware bietet Arduino auch eine Software zum Entwickeln von Programmcode an. Die IDE liefert die passenden Funktionen zum Programmieren der Boards und um deren Kommunikation über die USB-Schnittstelle zu untersuchen (serielle Schnittstelle). Ein übersichtliches Interface bietet Zugriff auf viele Code-Beispiele, welche vorinstalliert sind, sowie auf Bibliotheken, mit denen man die Standardfunktionen der Software erweitern kann. Die Bibliotheken stammen oft von Anbietern von Erweiterungshardware wie *LED Strips* oder *Displays* und bieten die Möglichkeit diese Hardware mit Arduino-Boards einzusetzen.
Eine große Rolle in dieser Bachelorarbeit spielt dabei eine Bibliothek zur Steuerung der Bildmi-

[3] Üblicherweise müssen Mikrocontroller in eine spezielle Hardware zum Programmieren gesteckt werden (den *Programmer*), um danach auf ihrer Platine eingebaut zu werden

scher über einen Arduino Mikrocontroller. Diese wird in Kapitel 3.3 näher beleuchtet.

Von großem Vorteil ist auch die internationale Entwickler-Community, welche mit Arduino Hard- und Software arbeitet. Dadurch kann auf einen umfassenden Pool an Informationen, Fragen und Antworten zurückgegriffen werden (siehe beispielsweise [Arduino Forum]).
Durch frühere Projekte mit Arduino Boards stand die Nutzung der Arduino Plattform von Anfang an fest. Die flexible Nutzung und schnelle Prototypisierung mit den Entwickler-Boards ist ein großes Argument für die Nutzung derselben.

## 2.4. Wireshark

Zur Analyse von Netzwerkverkehr, wie u.a. im Kapitel 3.2 angewandt, wird die Software *Wireshark* genutzt. Diese bietet die Möglichkeit auf OSI-Layer 1 (siehe Tabelle 2.2) Pakete im Netzwerk zu untersuchen. Dabei wird der Datenverkehr im lokalen Netzwerk mitgeschnitten und bei Bedarf über leistungsstarke Filter nach Protokollen sortiert.

| # | **Schicht/Layer** | **Beispiele** |
|---|---|---|
| 7 | Anwendungsschicht | HTTP, FTP, DNS, DHCP |
| 6 | Darstellungsschicht | |
| 5 | Sitzungsschicht | |
| 4 | Transportschicht | TCP, UDP |
| 3 | Vermittlungsschicht | IP, ICMP |
| 2 | Sicherungsschicht | Ethernet |
| 1 | Bitübertragungsschicht | |

Tabelle 2.2.: Übersicht über die Schichten des OSI-Modells

Das OSI-Schichtenmodell verdeutlicht die Kommunikation technischer Systeme. In Schicht eins findet die Übertragung im Medium statt, dies könnte z.B. ein digitales Signal in einem Kupferkabel sein. Schicht zwei bildet Rahmen aus den Signalen und hat die Funktion der Fehlererkennung in der Übertragung. Die dritte Schicht ordnet der Übertragung IP-Adressen zu, während Schicht vier Protokolle wie UDP und TCP implementiert [OSI Modell].
Die Pakete werden mit Wireshark auf Schicht eins gesammelt. Dadurch lassen sich einem solchen Paket viele Informationen entnehmen. Sowohl der Inhalt (die Daten) als auch alle Metadaten wie Sender, Empfänger und Timestamps können ausgelesen werden. Dies bietet einen sehr umfangreichen Spielraum bei der Analyse.

# 3. Analyse

Das Kapitel Analyse beschäftigt sich mit der eingehenden Untersuchung der Audio-Engine in ATEM-Bildmischer, sowie des Kommunikationsprotokolls und der Arduino Bibliothek. Ich beleuchte hier die Eigenschaften des eingebauten Audiomischers und den Zusammenhang mit den benötigten Befehlen. Durch Paketanalyse zeige ich den Aufbau eines vom Client an einen Bildmischer gesendeten Netzwerkpakets sowie die wichtigsten Befehle zu Steuerung des Bildmischers mithilfe des Controllers. Auch die Möglichkeiten und Grenzen der verwendeten Arduino Bibliothek werden aufgezeigt und analysiert.

## 3.1. ATEM Fairlight Audio

In allen ATEM-Bildmischern ist auch eine Audio-Engine[1] implementiert. Diese verfügt über eine vollwertiges Mischpult auf digitaler Basis und oft über digitales sound processing wie Equalizer oder Kompressoren. In früheren Versionen der ATEM-Bildmischern wurde das Audio über eine generische Engine gesteuert. In neueren Modellen wird dafür der *Fairlight Audiomixer* verwendet. Diese Engine ist auch in *DaVinci Resolve*, dem Videoschnittprogramm, implementiert. Vermutlich wurde Fairlight in die ATEM-Mischer integriert um eine einheitliche Audio-Engine über die verschiedenen Geräten zu schaffen.

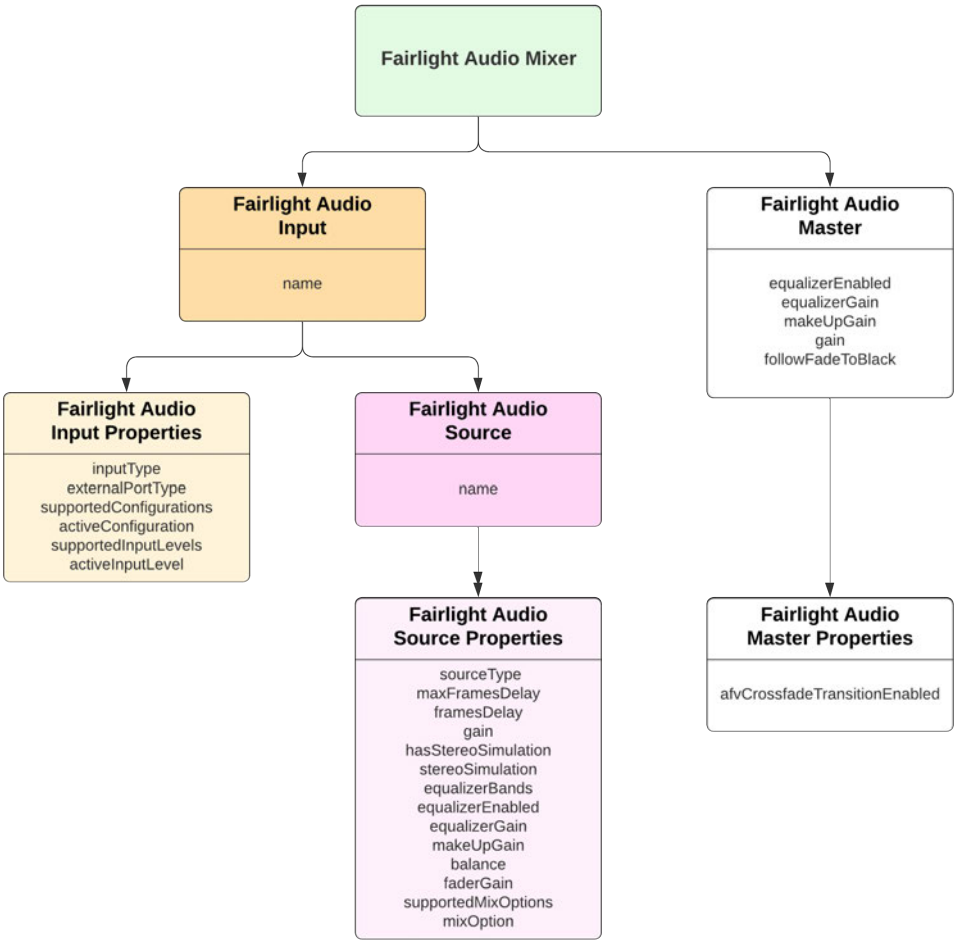
| **Input** | **input name** | **Source** (dez) | **Source** (hex) |
|---:|:---:|:---:|:---|
| HDMI 1 | 1 | -65280 | 0xFFFFFFFFFFFF0100 |
| HDMI 2 | 2 | -65280 | 0xFFFFFFFFFFFF0100 |
| HDMI 3 | 3 | -65280 | 0xFFFFFFFFFFFF0100 |
| HDMI 4 | 4 | -65280 | 0xFFFFFFFFFFFF0100 |
| Mic 1 (stereo) | 1301 | -65280 | 0xFFFFFFFFFFFF0100 |
| Mic 1 (mono links) | 1301 | -256 | 0xFFFFFFFFFFFFFF00 |
| Mic 1 (mono rechts) | 1301 | -255 | 0xFFFFFFFFFFFFFF01 |
| Mic 2 (stereo) | 1302 | -65280 | 0xFFFFFFFFFFFF0100 |
| Mic 2 (mono links) | 1302 | -256 | 0xFFFFFFFFFFFFFF00 |
| Mic 2 (mono rechts) | 1302 | -255 | 0xFFFFFFFFFFFFFF01 |

Tabelle 3.1.: Fairlight Struktur im ATEM Mini Pro mit ausgelesenen Werten

Die Struktur von Fairlight gleicht dem von Klassen bei der objektorientierten Programmierung. Der Übersicht halber habe ich sie als Organigramm (in Anlehnung an ein UML-Klassendiagramm)

[1] Mit Audio-Engine ist die Funktionalität gemeint Audio digital zu mischen und zu verarbeiten. An eine *Game-Engine* angelehnt verwende ich hier diesen Begriff

dargestellt (siehe Abbildung 3.1). Ein Eingang (*Fairlight Audio Input*) im Fairlight Mixer hat auch immer eine Quelle (*Fairlight Audio Source*). Beide haben Eigenschaften zugewiesen (*Properties*). Dieses Konzept muss beim Senden von Befehlen beachtet werden, da man sowohl einen Input steuern, als auch Quelleneigenschaften ändern kann.




Quelle: Eigene Darstellung

Abbildung 3.1.: Übersicht über die Struktur des Fairlight Audiomixers

Die Eingänge des Fairlight Mixers, welche direkt aus Kameraeingängen kommen, sind fortlaufend nummeriert (im ATEM Mini Pro 1...4 bei größeren Mischern entsprechend höhere Zahlen). Die beiden Mikrofoneingänge tragen die Nummer bzw. den Namen `1301` und `1302`. Während eine Quelle üblicherweise aus dem betreffenden Eingang entspringt, gibt es bei den Mikrofoneingängen die Möglichkeit zwischen einer Stereoquelle, z.B. für Signale aus einem externen Mischpult, und zwei Monoquellen umzuschalten. Die Stereoquelle trägt dann die Nummer `-65280`, während die Monoquelle aus dem linken Kanal die `-256` trägt und die des rechten Kanals `-255`. In Tabelle 3.1 sind alle Eingänge und Quellen des ATEM Mini Pro aufgeführt.

## 3.2. ATEM Kommunikationsprotokoll

Die ATEM Bildmischer kommunizieren über ein proprietäres Protokoll, welches vom Hersteller *Blackmagic* entwickelt wurde. Alle Bildmischer, vom aktuell kleinsten Modell, dem *ATEM Mini pro*, bis zu großen Modellen, wie dem *ATEM 4 M/E Broadcast Studio 4K*, können über das Netzwerk mit anderen Geräten kommunizieren. Dabei nutzen sie das *User Datagram Protocol (UDP)*, ein verbindungsloses und ungesichertes Protokoll aus der OSI-Layer 4. Das verbindungsorientierte Äquivalent ist das *Transmission Control Protocol (TCP)*, ebenfalls aus der Transportschicht [UDP].

Das ATEM Protokoll dient im Ökosystem der ATEM Mischer der Steuerung von Bildmischern und Kameras. Innerhalb eines Bildmischers können sämtliche Funktionen über das Protokoll ferngesteuert werden. Das macht es möglich, den Bildmischer vom eigentlichen Bedienteil zu separieren und so z.B. eine räumliche Trennung herzustellen. So müssen nicht sämtliche Kamerasignale per Kabel zur Regie bzw. Live-Cutter:in geführt werden, sondern können zentral in einem Rack zusammengeführt werden, was den Verkabelungsaufwand deutlich verringert. Eine solche Lösung bietet sich beispielsweise in einem Ü-Wagen an. Der Bildmischer selbst wird in einem Rack mit Zugang von außen montiert und wird über das Netzwerk mit einem Bedienteil (siehe Abbildung 2.2 in Kapitel 2.1) ferngesteuert.

Neben den vollständigen Bildmischer-Funktionen lassen sich auch Kameras von *Blackmagic* fernsteuern, wie z.B. die *Blackmagic URSA Broadcast G2* als professionelle Kamera für den Live-Einsatz oder aber auch kleinere (und kostengünstigere) Kameras wie die *Blackmagic Studio Camera*. Bei diesen (und weiteren) Modellen können die Kameraparameter aus der Ferne geändert werden, was einen professionellen Workflow mit Bildingenieur:innen[2] ermöglicht. So kann z.B. die Blende, der Weißabgleich oder Farbkorrekturen direkt von der Live-Cutter:in oder einer weiteren Person über den Bildmischer, eine Software oder sogenannte cameracontrol-Bedienteile gesteuert werden. Hierbei kommunizieren die Geräte ebenfalls über das ATEM Protokoll.

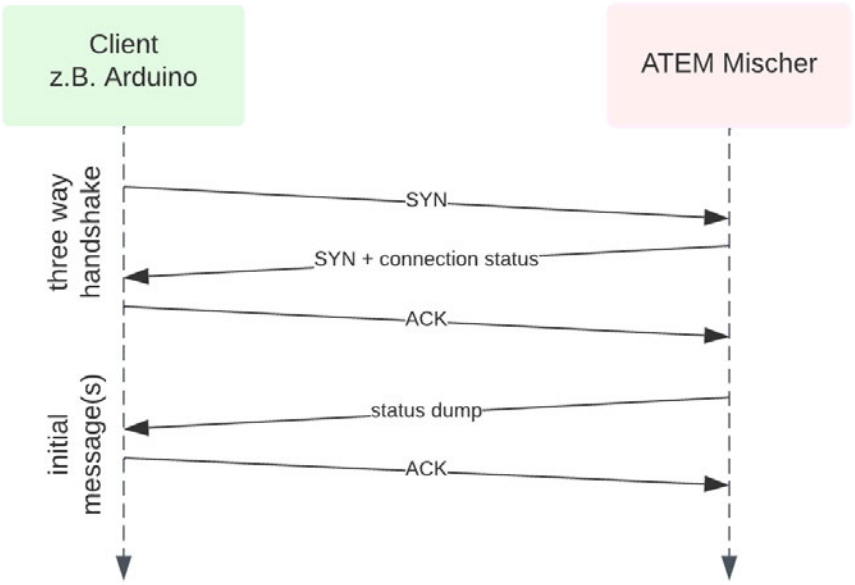
Das ATEM Protokoll bedient sich des grundlegenden Mechanismus des UDP, Datenpakete über das Netzwerk zu verschicken. Dabei operiert es auf Schicht vier im OSI-Schichtenmodell (der Transport-Ebene), wobei es die IP-Adresse nutzt, um ohne vorherigen Verbindungsaufbau[3] ein Paket an einen Empfänger zu senden [OSI Modell]. Durch die fehlende Kontrollstruktur im UDP, verfügt das ATEM Protokoll jedoch über eigene Mechanismen zur Verbindungssteuerung. So wurden „SYN“- und „ACK“-Pakete implementiert, welche im TCP nativ vorkommen und den Zweck haben eine Verbindung zu initiieren (SYN - syncronize (engl.) - Synchronisation) und den Empfang von Paketen bzw. Befehlen zu bestätigen (ACK - acknowledgement (engl.) - Bestätigung) [Open Switcher].

Zunächst baut ein Client (z.B. ein PC mit *ATEM Software Control*) eine Verbindung zu einem ATEM Mischer auf, indem es ein Initialisierungspaket (mit SYN-Flag[4]) sendet. Der Mischer ant-

[2] Die Bildingenieur:in bedient in einem Ü-Wagen z.B. die cameracontrol unit und greift damit aus der Ferne auf Kameraparameter zu, ohne dass die Kamera-Operator abgelenkt werden

[3] kein Verbindungsaufbau im Sinne des TCP; das ATEM Protokoll nutzt einen eigenen Verbindungsaufbau

[4] Als Flag bezeichnet man Statusindikatoren - hier wird die Bereitschaft zum Verbindungsaufbau signalisiert; Analogie zur Flagge am Strand (Schwimmen erlaubt/nicht erlaubt)




Quelle: Eigene Darstellung

Abbildung 3.2.: Verbindungsaufbau zwischen Client und ATEM Mischer als Sequenzdiagramm

wortet daraufhin ebenfalls mit einer SYN-Flag und dem Verbindungsstatus. Als letzter Schritt wird ein ACK-Paket vom Client gesendet, auf das der Mischer dann seine aktuelle Konfiguration bzw. seinen aktuellen Status in mehreren Paketen an den Client schickt (Siehe Abbildung 3.2) [Open Switcher]. Von jetzt an wird vom Client jedes Paket mit einem ACK-Paket bestätigt, andernfalls schickt der Mischer das Paket erneut, in der Annahme, der Client hat das Paket verloren oder nicht erhalten.

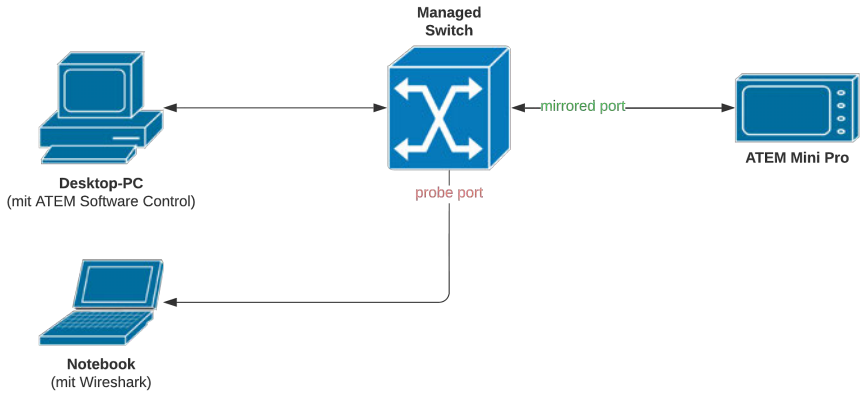
### 3.2.1. Analyse des Protokolls

Das ATEM Protokoll wurde von mir auf Netzwerkebene analysiert. Das bedeutet, ich habe mir den Netzwerkverkehr auf der Transportebene angesehen. So konnten die UDP Pakete direkt betrachtet werden. Zur Analyse wurde die Software *Wireshark* verwendet. Sie bietet ein übersichtliches grafisches Interface und kann schnell und einfach bedient werden.

Zunächst habe ich ein Testsetup (siehe Abbildung 3.3) aufgebaut, bestehend aus einem *ATEM Mini Pro*, einem Desktop-PC mit der *ATEM Software Control* Software und einem Notebook, auf dem Wireshark läuft. Der Bildmischer kommuniziert über einen „managed switch“[5] mit dem PC, das Notebook ist ebenfalls an diesem Switch angeschlossen. Im Switch wurden die Ports des PCs und des Notebooks via „port-mirroring“, also der Spiegelung der beiden Ports, konfiguriert. So werden alle vom PC ein- und ausgehenden Pakete an den Port des Notebooks gespiegelt und können von Wireshark empfangen und analysiert werden. Innerhalb des Switches werden diese Ports als „mirrored“- und „probe“-port bezeichnet.

[5] Ein „managed switch“ verfügt über anpassbare Einstellungen, wie z.B. VLAN oder port-mirroring

Der Paketverkehr fließt je nach Event vom Bildmischer zum Client oder umgekehrt. In letzterem Fall antwortet der Bildmischer immer mit dem aktualisierten Status des betreffenden Events. Sollte also vom Client aus z.B. der Befehl zum Schneiden gesendet werden, so geht ein Paket vom Client aus an den Mischer, welches den Schnittbefehl enthält und ein Paket (oft mehrere Pakete) vom Mischer zum Client, um den aktualisierten Status zu übermitteln. In diesem Beispiel zählt dazu der Programm-Input, der Preview-Input sowie Tally-Informationen.




Quelle: Eigene Darstellung

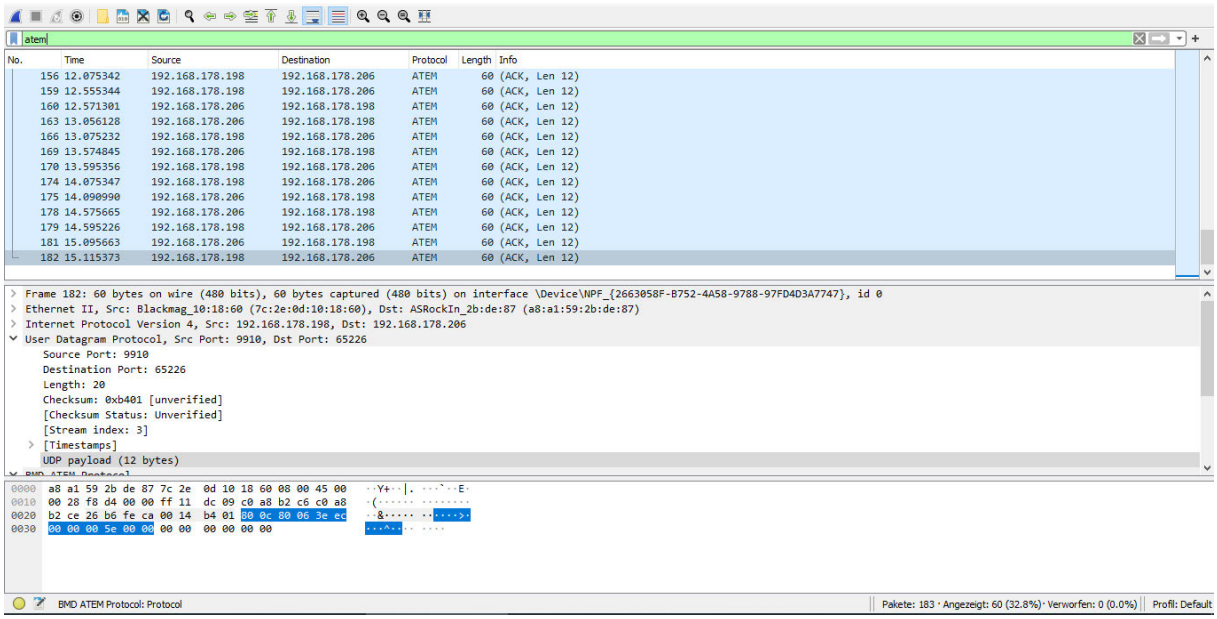
Abbildung 3.3.: Netzwerkdiagramm des Testaufbaus

Zunächst habe ich den reinen payload[6] der UDP Pakete analysiert. In der Regel erhält man so einen HEX-Dump des Pakets. Damit ist die Darstellung der Paketdaten im hexadezimalen Format gemeint, was eine Analyse der einzelnen Bytes vereinfacht. Will man sich ein Byte genauer ansehen, kann man Wireshark auch auf binäre Ansicht umschalten. So lassen sich einzelne Bits direkt auswerten. Später kam bei der Untersuchung der Pakete ein „Wireshark dissector“ hinzu. Dies ist eine Erweiterung für Wireshark, welche in meinem Fall auf das ATEM Protokoll angepasst wurde und es erlaubt nach ATEM-Paketen zu filtern, einige Befehle direkt auszuwerten und generell eine gute Übersicht über ein ATEM-Paket zu erhalten. Schnell fiel mir auf, dass die Pakete mit einer sehr hohen Frequenz von 4 Paketen pro Sekunde empfangen wurden, obwohl kein Befehl gesendet wurde (siehe Abbildung 3.4). Diese keepalive-Pakete werden von Client und Mischer ausgetauscht und sorgen für ein Aufrechterhalten der Verbindung. Für die Analyse einzelner Befehle in einem Paket habe ich die keepalive-Pakete jedoch anhand ihrer Länge gefiltert, da sie lediglich 12 Byte lang sind.

### 3.2.2. Aufbau der UDP-Pakete

Mit Hilfe der Basisklasse der Arduino-Bibliothek (siehe Kapitel 3.3) konnte ich mir einen ersten Überblick über die Zusammensetzung eines Paketes verschaffen. Durch Analyse des Codes konnte ich Rückschlüsse auf den Aufbau eines Paketes ziehen. Nachfolgend erläutere ich die wichtigsten Erkenntnisse.

[6] Mit payload sind die Nutzdaten in einem Netzwerkpaket gemeint




Quelle: Eigene Darstellung

Abbildung 3.4.: Wireshark Interface mit eine Liste von ungefilterten empfangenen Paketen

Soll ein Befehl von einem Client getriggert werden, schickt dieser ein Paket an den ATEM Mischer. Dieses Paket besteht aus einem *Header*, einem *CommandString*, den Daten des Befehls und einigen weiteren Feldern. Einige Pakete bestehen nur aus dem Header, in der Regel sind dies keepalive- oder Initialisierungspakete. Der Header umfasst 12 Byte, der CommandString 4 Byte. Die restlichen Daten sind auf 44 Byte und mehr untergebracht (siehe Tabelle 3.2). Relevant sind für meine Untersuchungen lediglich die Felder CommandString und CommandData. Die Bezeichnungen stammen dabei von mir, da es keine offiziellen Dokumentationen zur Verteilung der Bytes eines Paketes gibt. Einige Projekte, welche ebenfalls das Protokoll analysiert haben, nutzen eine ähnliche Bezeichnung (siehe auch [Open Switcher] und [SKAARHOJ Protocol]).

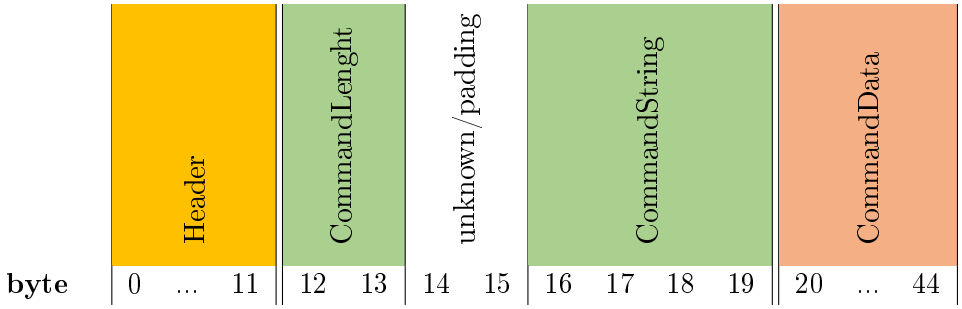



Tabelle 3.2.: Payload eines ATEM UDP Pakets

Der CommandString besteht aus 4 ASCII-Zeichen[7] und stellt eine Kurzform (oder Kurzbezeichnung) des Befehls dar. Die entsprechenden Langnamen konnte ich durch Recherche in verschiedenen ähnlichen Projekten ermitteln (mehr zu diesen in Kapitel 3.4) sowie durch Ableitungen aus den Kurzbezeichnungen.

[7] American Standard Code for Information Interchange

### 3.2.3. Relevante Befehle

Für den Betrieb des Controllers mit den erforderlichen Funktionen (siehe Kapitel 2.2) sind lediglich wenige Befehle notwendig. Die Audio-Steuerung ist dabei die Kernfunktion. Diese Befehle wurden von mir durch Paketanalyse ermittelt, indem ich in der ATEM Software auf dem PC einen Befehl ausgelöst habe, z.B. das Live-Schalten eines Audiokanals, und die daraufhin gesendeten Pakete analysiert habe.

| **Befehl** | **CommandString** | **Sender** | **Beschreibung** |
|---|---|---|---|
| Change Fairlight Source Properties | CFSP | Client | Ändert die Eigenschaften einer Fairlight Input Source |
| Fairlight Audio Source Properties | FASP | Mischer | Liefert Eigenschaften der Fairlight Input Source |
| Fairlight Mixer Tally | FMTl | Mischer | Gibt Auskunft über alle Fairlight Mixer Tallys |

Tabelle 3.3.: Relevante Befehle des ATEM Protokolls

Die in Tabelle 3.3 aufgeführten Befehle sind die relevanten Befehle zur Steuerung des Bildmischers durch den Controller. Der *Change Fairlight Source Properties*-Befehl ändert in einem Kanal unter anderem die Eingangsverstärkung (Gain), die Fader-Lautstärke, die Balance und das Audio-Delay. Auch kann man mit ihm den Kanal live schalten oder Audio-Follows-Video (AFV) aktivieren. Mit dem *Fairlight Audio Source Properties*-Befehl übermittelt der Bildmischer Änderungen an einer *Fairlight Audio Source*. Tally Informationen werden vom *Fairlight Mixer Tally*-Befehl an den Client gesendet.

In Tabelle 3.4 erkennt man die Felder oder Werte des `CFSP`-Befehls, die zugehörige Stelle in Byte und ihre Länge. Einige Bytes sind nicht mit einem Wert belegt, sondern werden als Puffer oder Padding freigelassen. Der zugehörige Datentyp gibt Aufschluss über mögliche Werte und definiert den Wertebereich.

Eine Besonderheit gibt es jedoch. Das Feld `changesFlags` markiert die Änderungen, welche mit dem Befehl vorgenommen werden. Da in der Regel nur eine Eigenschaft zur selben Zeit geändert wird (Fader wird bewegt, Mute wird gedrückt), wird immer ein Befehl für eine Eigenschaftsänderung gesendet und somit immer die zu ändernde Eigenschaft mithilfe der changesFlags markiert. Tabelle 3.4 zeigt auch die zugehörigen changesFlags für jedes Feld.

Der Aufbau des Befehls *Fairlight Audio Source Properties (FASP)* ist nahezu identisch mit dem zuvor beschriebenen `CFSP`. Während letzterer die Aufgabe hat, Änderungen an den Eigenschaften einer Fairlight Source zu veranlassen, übermittelt der `FASP` Befehl lediglich die Änderungen an den Client. Dies geschieht immer, wenn Änderungen an einer Audio Source vorgenommen werden, ob vom Bildmischer selbst (also auf dem Gerät) oder vom Client aus. Man kann diesen Befehl daher auch als Bestätigung für die erfolgreiche Änderung einer Eigenschaft ansehen.

| property | Byte | Länge | Datentyp | **Flags** (dez) |
|---|---|---|---|---|
| changesFlags | 0 | 2 | | |
| fairlightInput | 2 | 2 | | |
| padding | 4 | 4 | | |
| fairlightSource | 8 | 8 | int64 | |
| framesDelay | 16 | 1 | uint8 | 1 |
| padding | 17 | 3 | | |
| gain | 20 | 4 | int32 | 2 |
| stereoSimulation | 24 | 2 | uint16 | 4 |
| eqEnabled | 26 | 1 | bool | 8 |
| padding | 27 | 1 | | |
| eqGain | 28 | 4 | int32 | 16 |
| makeUpGain | 32 | 4 | int32 | 32 |
| balance | 36 | 2 | int16 | 64 |
| padding | 38 | 2 | | |
| faderGain | 40 | 4 | int32 | 128 |
| mixOption | 44 | 1 | uint8 | 256 |

Tabelle 3.4.: CommandData des CFSP Befehls

In erster Linie fällt auf, dass keine `changesFlags` übermittelt werden. Das liegt auch nahe, da hier keine Änderungen vorgenommen werden sollen, sondern lediglich ein Statusupdate übermittelt wird. Dann gibt es ein paar zusätzliche Felder, welche im `CFSP` nicht vorkommen. Dazu zählt z.B. `maxFramesDelay` also die maximale Anzahl an Frames, die das Audiosignal verzögert werden kann. Diese Informationen sind sinnvoll, wenn sich ein Client das erste Mal mit dem Mischer verbindet und seine Einstellungen mit denen des Mischers abgleichen möchte. So können z.B. Regler auf diesen Maximalwert eingestellt werden.

Abgesehen von einigen Verschiebungen in der Reihenfolge der Felder und anderer Verteilung der Padding-Bytes, sind die Befehle sehr ähnlich. Dies macht die Auswertung im späteren Parsing in Kapitel 4.1.2 einfacher. Die Unterschiede sind in der Tabelle 3.5 aufgeführt.

Ein weiterer relevanter Befehl ist *Fairlight Mixer Tally (FMTl)*. Seine Aufgabe ist die Mitteilung von Änderungen an den Tally-Lights im Fairlight Mixer. Diese Tallys kann man in der Software Control oberhalb der Kanäle sehen (siehe Abbildung 2.3), oder auf dem ATEM Mini Pro anhand der leuchtenden Buttons im Audio-Bereich (siehe Abbildung 2.1). Der Befehl besteht aus deutlich weniger Feldern als die beiden vorher beschriebenen, jedoch wiederholen sich diese innerhalb des Befehls. Zunächst gibt der Befehl in Byte `1` und `2` die Anzahl der im Befehl folgenden Tallys an. Darauf folgt eine Padding-Lücke von `6` Bytes. Die folgenden `8` Bytes beschreiben die Fairlight Audio Source und die Bytes `16` und `17` den Input, an dem die Quelle anliegt. Die Tally-Info liegt dann abschließend auf Byte `18`. Nun wiederholen sich die Felder um die zuvor im ersten Feld angegebene Anzahl für jeden weiteren Input und Quelle des Bildmischers. In der Regel ist dem Input nur eine Quelle zugeordnet, weshalb die Angabe des Inputs ausreichen würde. Sollte ein Kanal jedoch auf zwei Monoquellen aufgeteilt worden sein, können unterschiedliche Tally-Infos

für einen Kanal vorliegen. In diesem Fall benötigt man die Source-Angabe.

| property | Byte | Länge | Datentyp |
|---|---|---|---|
| fairlightInput | 0 | 2 | |
| padding | 2 | 8 | |
| fairlightSource | 8 | 8 | int64 |
| sourceType | 16 | 1 | uint8 |
| maxFramesDelay | 17 | 1 | uint8 |
| framesDelay | 18 | 1 | uint8 |
| padding | 19 | 1 | |
| gain | 20 | 4 | int32 |
| hasStereoSimulation | 24 | 1 | bool |
| padding | 25 | 1 | uint8 |
| stereoSimulation | 26 | 1 | uint8 |
| padding | 27 | 1 | |
| eqBands | 28 | 1 | uint8 |
| eqEnabled | 29 | 1 | bool |
| padding | 30 | 2 | |
| eqGain | 32 | 4 | int32 |
| makeUpGain | 36 | 4 | int32 |
| balance | 40 | 2 | int16 |
| padding | 42 | 2 | |
| faderGain | 44 | 4 | int32 |
| supportedmixOptions | 48 | 1 | uint8 |
| mixOption | 49 | 1 | uint8 |

Tabelle 3.5.: CommandData des FASP Befehls

| property | Byte | Länge | Datentyp |
|---|---|---|---|
| countTallys | 0 | 2 | uint16 |
| padding | 3 | 6 | |
| fairlightSource | 8 | 8 | int64 |
| fairlightInput | 16 | 2 | uint16 |
| tally | 18 | 1 | bool |

Tabelle 3.6.: CommandData des FMTl Befehls

## 3.3. Arduino Bibliothek

Ein Großteil dieser Bachelorarbeit baut auf der *Arduino ATEM library* von *SKAARHOJ ApS*[8] (nachfolgend auch SKAARHOJ) auf. Diese Bibliothek für die Arduino-Entwicklungsumgebung ist ein Produkt von *reverse engineering* des Kommunikationsprotokolls der ATEM Bildmischer und geht auf den Entwickler *Kasper Skårhøj* zurück [SKAARHOJ Protocol]. Sie wurde unter der *GNU General Public License (GNU GPL) Version 3* als „open source" veröffentlicht und kann daher frei verwendet, verändert, verbreitet, etc. werden [GNU GPL v3].

Die Funktion der Bibliothek ist eine Verbindung zu den ATEM Bildmischern aufzubauen, diese zu steuern sowie deren Status abzufragen. Dazu zählen sämtliche Funktionen vom Schneiden der Kamerakanäle bis zum Aktivieren von Keyern oder anderen Effekten. Auch die Audio-Funktionen der Mischer können so extern gesteuert werden. Durch verschiedene Ausführungen der Bibliothek variiert der Umfang des Befehlssatzes, sodass sie für kleinere Projekte auch auf Hardware mit geringem Programmspeicher einsetzbar ist. Die Ausführungen teilen sich auf in

- ATEMbase
- ATEMmin
- ATEMstd
- ATEMmax
- ATEMext
- ATEMuni

wobei für den weiteren Verlauf der Bachelorarbeit lediglich die Version *ATEMbase* sowie *ATEMstd* relevant sind. Innerhalb der Versionen besteht die Bibliothek aus einer gleichnamigen Klasse. Während die Klasse *ATEMmax* über einen vollständigen Funktionsumfang verfügt, lassen sich mit kleineren Klassen wie *ATEMmin* nicht alle Funktionen fernsteuern.

*ATEMstd* verfügt über alle für diese Bachelorarbeit benötigen Funktionen (bzw. Methoden) und bietet einen guten trade-off zwischen Speicherplatz und verfügbarer features. Beim verwendeten *Arduino Mega* muss zwar nicht in erster Linie auf Speicherplatz geachtet werden, dennoch erhält man durch das Sparen von Flash-Memory einige Vorteile, wie z.B. kürzere Programmierzeiten („flashen" des Programms).

Die Klasse *ATEMbase* (im Folgenden auch Basisklasse genannt) besteht ausschließlich aus Methoden und Variablen, welche für den Verbindungsaufbau mit einem ATEM Mischer benötigt werden und wird in allen Subklassen, wie *ATEMstd*, eingebunden. Hier werden beim Initiieren der Klasse IP-Adresse und Port, welche für die Verbindung notwendig sind, übergeben und sorgen für die Verbindung mit dem richtigen Bildmischer. Die Basisklasse verfügt auch über eine *runLoop()* Methode, welche innerhalb der Arduino *loop()*-Funktion (mehr dazu in Kapitel 4.2) permanent aufgerufen werden muss. Dadurch kann auf eingehende Pakete reagiert werden und die Verbindung zum Bildmischer durch permanenten Paketaustausch aufrecht erhalten werden.

[8] SKAARHOJ ApS - ein dänischer Elektronikhersteller; ApS steht für *Anpartsselskab* und beschreibt die Rechtsform des Unternehmens

Seit August 2018 werden die Bibliotheken nicht mehr gewartet. Dadurch wird die Funktionalität vom Entwickler nur bis zur Firmware Version *7.5.0* garantiert. Darüber hinaus besteht nur eine Funktionsgarantie beim Kauf von kommerziellen Produkten des Herstellers *SKAARHOJ* [SKAARHOJ Protocol].

Diese Bibliotheken von *SKAARHOJ* wurden von mir bereits in einem vorherigen Projekt erfolgreich eingesetzt. Beim Bau eines Tally-Lights[9] konnte ich mit einem *WEMOS D1 mini* Entwickler-Board und der *ATEMstd* Bibliothek eine Verbindung zu einem beliebigen ATEM Bildmischer aufbauen und diese über einen langen Zeitraum aufrecht erhalten (bei einigen Tests waren es über 3 Stunden). Des Weiteren gelang es mir die Tally Informationen pro Kamera-Kanal auszulesen und mittels einer LED so ein Rotlicht für eine Kamera nachzurüsten. Da ich diese Tally-Lights mit mehreren aktuellen ATEM Mischern testen konnte (unter anderem mit dem *ATEM Mini pro*), kann ich bestätigen, dass die Bibliothek auch mit aktueller Firmware der Bildmischer kommunizieren kann und die Verbindung erfolgreich aufgebaut wird. Jedoch ist der Funktionsumfang deutlich eingeschränkt. Vermutlich wurde die Firmware weiterentwickelt, sodass einige Befehle zu Steuerung des Bildmischers in der Bibliothek fehlen. So existieren z.B. keine Methoden zur Änderung der Eingangsverstärkung (Gain) eines am Bildmischer angeschlossenen Mikrofons oder zur Anpassung des Delays zwischen Bild und Ton pro Kanal. Diese Methoden gilt es in der Erweiterung der Befehlssatzes nachzurüsten, sodass alle benötigten Funktionen des Audio-Controllers über die Bibliothek abgebildet werden können.

### 3.3.1. Senden von Befehlen

Die Bibliothek besteht aus vielen Methoden, welche jeweils für einen Befehl des Bildmischers stehen. Oft wurden weitere Unterfunktionen angelegt, um die Befehle noch präziser zu senden. Jede dieser Methoden bildet den CommandData-Teil eines Paketes, sodass in der Controller-Software komfortabel über eine Funktion, z.B. die Lautstärke eines Kanals erhöht werden kann. Der folgende Code zeigt ein Beispiel aus der Klasse *ATEMstd*.

```
1  void ATEMstd::setAudioMixerInputVolume(uint16_t audioSource, uint16_t volume) {
2    _prepareCommandPacket(
3      PSTR("CAMI"),
4      12,
5      (_packetBuffer[12+_cBBO+4+4+2]==highByte(audioSource)) &&
6      (_packetBuffer[12+_cBBO+4+4+3]==lowByte(audioSource))
7    );
8    _packetBuffer[12+_cBBO+4+4+0] |= 2; // Set Mask: 2
9    _packetBuffer[12+_cBBO+4+4+2] = highByte(audioSource);
10   _packetBuffer[12+_cBBO+4+4+3] = lowByte(audioSource);
11   _packetBuffer[12+_cBBO+4+4+6] = highByte(volume);
12   _packetBuffer[12+_cBBO+4+4+7] = lowByte(volume);
13 }
```

Code 3.1: Beispiel zum Senden eines Befehls

Die Funktion `_prepareCommandPacket` bereitet das Paket mit passender CommandLength und dem dazugehörigen CommandString vor. Die Zeilen `_packetBuffer[]` setzen die Bytes der CommandData an die Stellen, an denen es der Bildmischer erwartet. Dieses Vorgehen basiert

[9] „Rotlicht“ auf einer Kamera, das den Live-Status dieser signalisiert

vermutlich ebenfalls auf Paketanalyse und dem darauffolgenden Nachahmen eines Paketes.

### 3.3.2. Empfangen von Befehlen/Paketen

Auch der Rückweg, also das parsen[10] eines empfangenen Paketes, wird in einer Methode vorgenommen. Die unterschiedlichen Klassen, wie z.B. *ATEMstd*, überladen dabei die Methode `_parseGetCommands()` in der Klasse *ATEMbase*. Ein (gekürztes) Beispiel aus *ATEMstd* zeigt die Vorgehensweise beim parsen.

```
1  if(!strcmp_P(cmdStr, PSTR("PrgI"))) {
2    mE = _packetBuffer[0];
3    if (mE<=1) {
4      atemProgramInputVideoSource[mE] = word(_packetBuffer[2], _packetBuffer[3]);
5    }
6  }
```

Code 3.2: Auszug aus der Funktion `_parseGetCommands()`

Zunächst wird der CommandString verglichen. Wenn er, wie in diesem Beispiel, `PrgI` entspricht, wird aus dem `_packetBuffer` an Stelle `0`, also dem ersten Byte der CommandData, der Wert der Mischerebene (ME)[11] ausgelesen. Die live geschaltete source wird dann aus Byte `2` und `3` in eine Variable gelesen. Mit diesem Befehl teilt der Bildmischer dem Client also mit, welcher Kanal (auf welcher Mischerebene) gerade live ist und der Client muss diese Informationen korrekt parsen.

Die Informationen, welche aus den Befehlen extrahiert wurden, werden dann in Variablen vorgehalten und können über sogenannte *getter-Methoden* (auch Zugriffsfunktion oder nur „getter“) abgefragt werden. Die getter sind neben den settern eine Kapselfunktion für private Variablen. Da diese nicht von außerhalb einer Klasse geändert oder gelesen werden können, benötigt man eine öffentliche getter- oder setter-Funktion
Das folgende Beispiel auf einem anderen Projekt (TallyLight) verdeutlicht das Nutzen dieser Informationen.

```
1  tallyStatus = AtemSwitcher.getTallyByIndexTallyFlags(cam-1);
2  if((tallyStatus == 1 || tallyStatus == 3) && !live) {
3
4    Serial.println("Kamera 1 ist LIVE!!");
5    live = true;
6    prev = false;
7
8    digitalWrite(D3, HIGH);
9    digitalWrite(D4, LOW);
10 }
```

Code 3.3: Beispiel zur Nutzung von Befehls-Daten nach dem Parsen

[10] Als „parsen“ bezeichnet man das zerlegen und analysieren von Daten
[11] Große Bildmischer verfügen oft über mehrere physische Ebenen, welche zum Mischen genutzt werden können

## 3.4. Ähnliche Projekte

Im Zuge der Recherchen zu dem ATEM Protokoll und der Arduino Bibliothek wurde ich auch auf weitere Projekte aufmerksam, welche das ATEM Protokoll analysiert haben. Allen voran steht das Projekt *LibAtem*, welches ebenfalls von der Arbeit von *SKARRHOJ* profitiert. Die Bibliothek ist in `C#` geschrieben und bietet eine nahezu vollständige auf reverse-engineering basierende Implementierung des ATEM Protokolls (siehe [LibAtem]). Aus dem Quellcode der Bibliothek konnte ich hilfreiche Informationen zur Struktur der Befehle gewinnen.

Auf der vorstehenden Bibliothek aufbauend ist das GitHub-Repository *sofie-atem-connection* des norwegischen Fernsehsenders *Norsk rikskringkasting (NRK)*. Diese Bibliothek ist eine Erweiterung für ihre TV Automations-Software *Sofie*. Diese Softwareerweiterung stellt eine Verbindung mit ATEM Bildmischern her und lässt zu, dass *Sofie* direkt Schnitte oder weitere Einstellungen auf dem Bildmischer vornimmt. Geschrieben wurde sowohl die Automations-Software als auch die Bibliothek in *Typescript* (siehe [NRK Sofie] und [Sofie ATEM connection]).
Die einfache Installation der Bibliothek macht schnelle Tests möglich und Analysen des Netzwerkverkehrs lassen Vergleiche der Pakete mit den eigenen Aufzeichnungen zu. Durch Ausgabe vom „state“ der Bibliothek, konnte ich den Fairlight Mixer tiefer gehend untersuchen und die unterschiedlichen Sources sowie Inputs beleuchten.

Gegen Ende meiner Recherchen wurde ich auf ein Projekt aufmerksam, was das ATEM Protokoll dokumentiert. Der Verbindungsaufbau wird hier detailliert beschrieben und bot mir die Möglichkeit, Informationen für die Analyse der Pakete zu gewinnen. Auch einige Befehle, welche den Fairlight Mixer steuern, werden beschrieben, die Liste ist jedoch nicht vollständig [Open Switcher].

# 4. Entwicklung

Die Entwicklung des Audio-Controllers teilt sich in die Erweiterung der Arduino-Bibliothek und das Design sowie die Hardwareentwicklung auf. Während die Weiterentwicklung der Arduino-Bibliothek zunächst ohne finales Hardwaredesign auskommt, benötigt der Controller, diverse Funktionen aus der erweiterten Bibliothek, sobald das Hardwarelayout steht.
Im Rahmen der Entwicklung habe ich daher zunächst die Bibliothek erweitert und danach das Hardwarelayout erstellt sowie die Controller Software geschrieben.

Da ein großer Teil der Arbeit schon in der Analyse des Protokolls und der Bibliothek passierte, gestaltete sich die Entwicklung selbst als verhältnismäßig einfach. Die Umsetzung des analysierten Protokolls in die Software war ein deutlich geringerer Zeitaufwand als das Untersuchen der einzelnen Befehle. Das Schreiben des Programmcodes für den Controller selbst, geschah in Teilen neben der Entwicklung der Hardware. Sobald ein Bauteil eingetroffen war, wurde es auf dem Protoboard eingebaut und getestet. Der aus diesen Tests entstandene Programmcode wurde teilweise mit einigen Optimierungen bis in den finalen Controller übernommen.

## 4.1. Erweiterung des Befehlssatzes der Bibliothek

Nachdem ich die Pakete, deren Aufbau und die Bibliothek selbst analysiert habe, konnte ich die Bibliothek um die fehlenden Funktionen bzw. Methoden erweitern. Ziel der Erweiterung war, alle relevanten Controller-Funktionen in Befehle und somit in Netzwerkpakete, welche an den Bildmischer gesendet werden, zu übersetzen. Da die Analyse die wichtigsten Befehle ergab, habe ich zunächst den *Change Fairlight Source Properties (CFSP)* implementiert.

Da die Bibliotheken von *SKAARHOJ* einer gewissen Namenskonvention folgen (siehe Kapitel 3.3), habe ich ebenfalls eine Bibliothek angelegt und ihr den Namen *ATEMpro* gegeben. Der Name ist zum einen angelehnt an den *ATEM Mini Pro*, zum anderen soll diese Bibliothek fehlende Funktionen ergänzen und damit professionelle Einsatzmöglichkeiten eröffnen. Die Bibliothek lässt sich wie eine der *SKAARHOJ* Bibliotheken (*ATEMmax*, *ATEMstd*) in ein Arduino-Projekt einbinden und besteht aus einer gleichnamigen Klasse. Diese ist, wie für Arduino Bibliotheken üblich, in `C++` geschrieben.

Innerhalb der *Header*-Datei werden zunächst alle privaten sowie öffentlichen Variablen und Methoden definiert. Hierzu zählen die Variablen, welche den Status des Mischers spiegeln, z.B. `faderGain[30]`. Diese Variable hält den Gain-Wert (die Position) des Faders für bis zu 30 verschiedene Kanäle. Auch die Methoden sind zunächst im Header definiert. `_parseGetCommands()` ist die wichtige Methode zum Parsen, `setFramesDelay()` ist ein Beispiel für eine vorgeschaltete Methode[1] für einen Befehl.

[1] So muss eine Methode mit vielen Parametern nicht direkt im Controller-Code aufgerufen werden

In der eigentlichen Bibliothek, der `.cpp`-Datei sind alle Deklarationen zu finden. Hier steht der Programmcode.

### 4.1.1. CFSP

Der Befehl *Change Fairlight Source Properties (CFSP)* ist der relevanteste, da er die Lautstärke eines Kanals, den Gain, sowie Mute (bzw. live schalten) ändern kann. Die benötigten Informationen zur Implementierung des Befehls in der Bibliothek finden sich in Kapitel 3.2.3. Zunächst habe ich den CommandHeader vorbereitet, indem ich die Funktion `_prepareCommandPacket` aus der ATEMbase Klasse nutze. Diese nimmt als Parameter den CommandString und die CommandLength in Bytes entgegen.

```
1 _prepareCommandPacket(
2   PSTR("CFSP"), // set CommandString
3   48    // set CommandLength in Byte
4 );
```

Code 4.1: Beginn der Methode des `CFSP` Befehls

Im weiteren Verlauf habe ich die Felder des Befehls den entsprechenden Bytes zugewiesen und mit den Werten gefüllt. Dies sieht im Code komplex aus, da hier die Option auf ein *CommandBundle* vorgesehen ist. Ein CommandBundle vereint zwei oder mehr Befehle in einem Paket und muss manuell gebildet werden. Über die Methode `commandBundleStart()` kann das Bundle begonnen und daraufhin mehrere Befehle erzeugt werden. Diese werden dann in ein Paket serialisiert und versendet, nachdem man das Bundle mit `commandBundleEnd()` geschlossen hat.
Der nachfolgende Codeabschnitt zeigt exemplarisch das Feld Gain, aufgeteilt auf die vier zur Verfügung stehenden Bytes.

```
1 _packetBuffer[12 + _cBBO + 4 + 4 + 20] = (gain >> 32) & 0xFF;
2 _packetBuffer[12 + _cBBO + 4 + 4 + 21] = (gain >> 16) & 0xFF;
3 _packetBuffer[12 + _cBBO + 4 + 4 + 22] = (gain >> 8) & 0xFF;
4 _packetBuffer[12 + _cBBO + 4 + 4 + 23] = gain & 0xFF;
```

Code 4.2: Aufteilen eines 32-Bit Integer-Wertes auf 4 Bytes

Das Array `_packetBuffer[]` hält die Bytes des Befehls vor, bis sie gemeinsam als UDP Paket gesendet werden. Im Index des Arrays findet eine kleine Addition statt, bestehend aus der `12` für die Bytes des CommandHeaders, ggf. einem CommandBunde-Offset, `4` Bytes für CommandLength (`2`) und Padding (`2`) und `4` Bytes für den CommandString. Weitere `20` Bytes weiter steht dann das MSB[2] des Gain-Feldes.
An dieser Stelle nutze ich die Bitverschiebung und Maskierung, um den Int32-Wert der `gain` Variable in seine Hexadezimaldarstellung zu zerlegen. `gain » 32` verschiebt die Bits der Variable um 32 Bit nach rechts, wo dann mit `& 0xFF` lediglich das letzte Byte (LSB) übernommen wird. Die nächste Zeile verschiebt dann die Variable um 16 Bit usw. So wird aus einer Int32-Zahl eine 4 Byte Hexadezimaldarstellung in der sogenannten *Big Endian* Ordnung (LSB ganz rechts).

[2] Most Significant Bit - das höchstwertigste Bit einer Zahl

Wenn alle Felder an der entsprechenden Stelle im `_packetBuffer` stehen, kann das Paket mit dem Befehl `_finishCommandPacket()` geschlossen werden. Damit wird der CommandHeader an den Anfang geschrieben und das Paket per UDP gesendet.

Zur Vereinfachung des Befehls habe ich mehrere Funktionen (oder präziser „Überfunktionen“) geschrieben. Diese werden von der Controller-Software aufgerufen und haben den Vorteil, dass Sie den Befehl deutlich vereinfachen. So müssen nicht alle Felder in Form von Parametern an die Funktion übergeben werden, sondern es wird eine übergeordnete Funktion aufgerufen, welche ihrerseits nur zu ändernde Parameter entgegennimmt. Dann wird die Funktion des `CFSP` Befehls aufgerufen und alle nicht übergebenen Parameter gleich null gesetzt. Ein Code-Beispiel verdeutlicht dieses Verfahren anhand des `changeFaderGain()` Befehls.

```
1  void ATEMpro::setFaderGain(uint16_t channel, int64_t source, float faderGain) {
2    int32_t fg = (faderGain * 100 - 90) * 100; // this gives a range from -90 to +10 (dB)
3    changeFairlightSourceProperties(128, channel, source, 0, 0, 0, 0, 0, 0, 0, fg, 0);
4  }
```

Code 4.3: Vorgeschaltete Funktion zur Vereinfachung der `CFSP` Methode

Nur die zum Ändern des Fader-Gains notwendigen Parameter (Input-Channel, Source, fader-Gain) werden von der Funktion abgefragt. Es wird die notwendige Flag für die Änderung des Fader-Gains gesetzt und die anderen Felder auf 0 gesetzt. Untersuchungen zeigten, dass es irrelevant ist, was in den Feldern steht, die nicht durch die changesFlag markiert sind, der Bildmischer ignoriert diese ohnehin. In diesem Fall wird noch eine Berechnung zur Erweiterung der dB-Werte durchgeführt, um auf eine Skala von `-90` bis `+10` zu gelangen.

### 4.1.2. FASP

Eine Art Gegenstück zum `CFSP` bietet der Befehl *Fairlight Audio Source Properties (FASP)*. Er wird vom Bildmischer an den Client gesendet und muss daher geparsed werden. In Kapitel 3.2.3 wurde der Befehl ausführlich beleuchtet und mithilfe der Tabelle 3.5 in die Felder und ihre Byte Position zerlegt. Daher ist das Vorgehen beim parsen relativ übersichtlich. Zunächst wird die Funktion `_parseGetCommands()` aus der *ATEMbase* Klasse innerhalb der neuen Bibliothek angelegt. Der Vorgang nennt sich Überladung und hat den Vorteil, unterschiedliche Parameter für Funktionen zu übergeben. Der Compiler[3] benutzt dann die jeweils benötigte Funktion beim Kompilieren.

```
1  void ATEMpro::_parseGetCommands(const char* cmdStr) {
2    _readToPacketBuffer();  // read all bytes from packet to buffer
3
4    // we get the command string via method-params so we can and should
5    // check for each command individually by comparing strings (if 0 -> they're equal)
6    if (!strcmp_P(cmdStr, PSTR("FASP"))) {
7      uint16_t channel = (uint16_t)word(_packetBuffer[0], _packetBuffer[1]);
8      uint8_t channelIndex= getAudioSrcIndex(channel);
9
10     maxFramesDelay[channelIndex] = (uint8_t)_packetBuffer[17];
11     framesDelay[channelIndex] = (uint8_t)_packetBuffer[18];
```

[3] C++ muss als Hochsprache in eine maschinenlesbare Sprache übersetzt werden, dies geschieht beim Kompilieren durch den Kompilierer oder Compiler

```
12      balance[channelIndex] = (uint16_t)word(_packetBuffer[40], _packetBuffer[41]);
13    }
14  }
```

Code 4.4: Ausschnitt aus dem Parsing des `FASP`

Im oberen Codeblock sieht man die Funktionsweise am Beispiel von `maxFramesDelay`, `framesDelay` und `balance`. Zunächst wird das Paket in den Buffer eingelesen. Dann vergleicht die Funktion `strcmp_P()` den CommandString mit dem angegebenen `FASP`. Gibt es keinen Unterschied zwischen den beiden Strings, wird eine `0` zurückgegeben. Innerhalb der `if`-Anweisung werden nun die Bytes aus dem Buffer mittels Typecasting in den entsprechenden Datentyp gewandelt und in die klassenweiten Variablen geschrieben.
Ist der (hier gekürzte) Programmcode durchlaufen, sind die Informationen aus dem Befehl in die Variablen übertragen worden und können abgerufen und genutzt werden.

### 4.1.3. FMTl

Der Befehl *Fairlight Mixer Tally (FMTl)* ist deutlich kürzer als die bisher beschriebenen. Er verfügt lediglich über fünf Felder zur Auswertung. Jedoch hat er die Besonderheit, dass er sich um eine zuvor in Feld 1 angegebene Anzahl wiederholt. Das Parsing sieht also vor, dass zunächst die Anzahl der Wiederholungen ausgelesen wird und dann mit einer Schleife alle Felder entsprechend der Anzahl der Wiederholungen ausgelesen werden.

```
1  if (!strcmp_P(cmdStr, PSTR("FMTl"))) {
2      uint16_t countTallys = (uint16_t)word(_packetBuffer[0], _packetBuffer[1]);
3
4      for(int i = 0 ; i < countTallys ; i++) {
5        uint16_t input = (uint16_t)word(_packetBuffer[16 + (i*11)], _packetBuffer[17 + (i*11)]);
6        uint16_t source = 0; //TODO
7        fairlightTally[getAudioSrcIndex(input)][getIndexByFairlightSource(source)] = (bool)word(
             _packetBuffer[18 + (i*11)]);
8      }
9    }
```

Code 4.5: Parsing des `FTMl` Befehls

Der Abstand zwischen den aneinandergereihten Befehlen beträgt `11` Byte. Sobald also die Schleife den ersten Durchlauf beendet hat, erhöht der Zähler `i` auf `1` und lässt z.B. den Ausdruck `_packetBuffer[18 + (i*11)]` zu `_packetBuffer[29]` werden. Damit steht die Position der nächsten Tally Information im Index des Buffers und kann ausgelesen werden.

## 4.2. Software des Controllers

Als Software des Controllers bezeichne ich den Programmcode, welcher auf das Arduino-Board programmiert wird und für die Funktion des Controllers selbst sorgt. Die Programmiersprache, die dafür verwendet wird, ist `C++` mit einigen Besonderheiten der Arduino Plattform. So kommen etwa Funktionen vor, welche speziell für die Arduino Hardware vorgesehen sind und in Standard-`C++` nicht vorkommen. Die Arduino-Sketches[4] werden beim Kompilieren mit der Arduino IDE einem Preprocessing unterzogen. Dieser Prozess bindet die Arduino-spezifischen

[4]Mit der Arduino IDE erstellte Programmcodes werden als `.ino`-Sketch abgespeichert

Funktionen über den Header `Arduino.h` ein. Während des Preprocessing werden auch alle Funktionen definiert. [Arduino Forum]

Für das Programmieren der Software verwende ich die IDE *Visual Studio Code* mit der Erweiterung *platform.io*. Letztere bietet die Möglichkeit Arduino-Boards außerhalb der Arduino IDE zu programmieren und verbindet damit die Arduino Plattform mit einer professionellen IDE. So kann ich von hilfreichen Funktionen, wie Code-Vervollständigung, Syntax-Highlighting und einem umfangreichen Interface profitieren. Auch habe ich so die Möglichkeit, die von mir erstellte Bibliothek direkt im Projektmappen-Explorer anzuzeigen und schnell zwischen Controller-Software und Bibliothek zu wechseln.

Ein typischer Arduino Programmcode besteht aus zwei Hauptfunktionen. Zum einen der *Setup-Routine* und zum anderen der *Main-Loop*. Die Setup Funktion wird beim Starten des Mikrocontrollers einmalig ausgeführt und bietet die Möglichkeit hier Programmcode zu schreiben, der eine initiierende Funktion hat. So nutze ich diese Routine um alle Komponenten, wie das Display, die Ethernet-Verbindung oder den mDNS-Service zu starten. Auch werden alle benötigten Pin-Modi[5] eingestellt, die Interrupts verbunden und eine serielle Verbindung gestartet.

```
1  void setup() {
2    Serial.begin(9600);
3    randomSeed(analogRead(A5));
4
5    // Init Display
6    if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) { die(); }
7    display.clearDisplay();
8    display.setCursor(0, 0);
9    display.setTextSize(1);
10   display.setTextColor(WHITE);
11   display.println("Display... OK");
12   display.display();
13
14   // Init Ethernet
15   display.print("Starte Ethernet... ");
16   display.display();
17
18   Ethernet.begin(mac, ip);
19   display.print("OK\n");
20
21   // Init mDNS
22   display.print("Starte mDNS... ");
23   display.display();
24
25   EthernetBonjour.begin("arduino");
26   EthernetBonjour.setServiceFoundCallback(ATEMfound);
27   display.print("OK\n");
28
29   // Encoder Setup
30   pinMode(debugLed, OUTPUT);
31   pinMode(2, INPUT_PULLUP);
32   pinMode(3, INPUT_PULLUP);
33   pinMode(5, INPUT_PULLUP);
34   attachInterrupt(digitalPinToInterrupt(2), encInterrupt, CHANGE);
```

[5] Der Arduino bietet die Möglichkeit per Software digitale Ports von Ein- auf Ausgang umzuschalten, oder einen integrierten pull-up Widerstand einzuschalten

```
35    attachInterrupt(digitalPinToInterrupt(3), encBtnInterrupt, CHANGE);
36
37    delay(500);
38  }
```

Code 4.6: setup-Routine der Controller-Software in `main.cpp`

Die Loop-Funktion wiederholt sich im Betrieb des Mikrocontrollers permanent. Sie wird mit der Taktgeschwindigkeit des Prozessors wiederholt (was in meinem Fall 16 MHz wären), vorausgesetzt sie benötigt keine Zeit zum Durchlaufen. In der Praxis kommt man auf deutlich niedrigere Werte. Innerhalb der Funktion wird der eigentliche Programmcode durchlaufen. Funktionen zum Parsen von Paketen werden aufgerufen, Steuerelemente werden abgefragt und das Display beschrieben.

```
1   void loop() {
2     if(systemState == 0) {
3       ...
4       EthernetBonjour.run();
5       menu();
6       ...
7     }
8     if(systemState == 1)
9     {
10      display.clearDisplay();
11      display.setCursor(0, 0);
12      display.println("Verbinde mit");
13      display.print(foundAtemAddresses[menuPosition][0]);
14      display.print(".");
15      display.print(foundAtemAddresses[menuPosition][1]);
16      display.print(".");
17      display.print(foundAtemAddresses[menuPosition][2]);
18      display.print(".");
19      display.print(foundAtemAddresses[menuPosition][3]);
20      display.display();
21
22      connectToATEM();
23    }
24  }
```

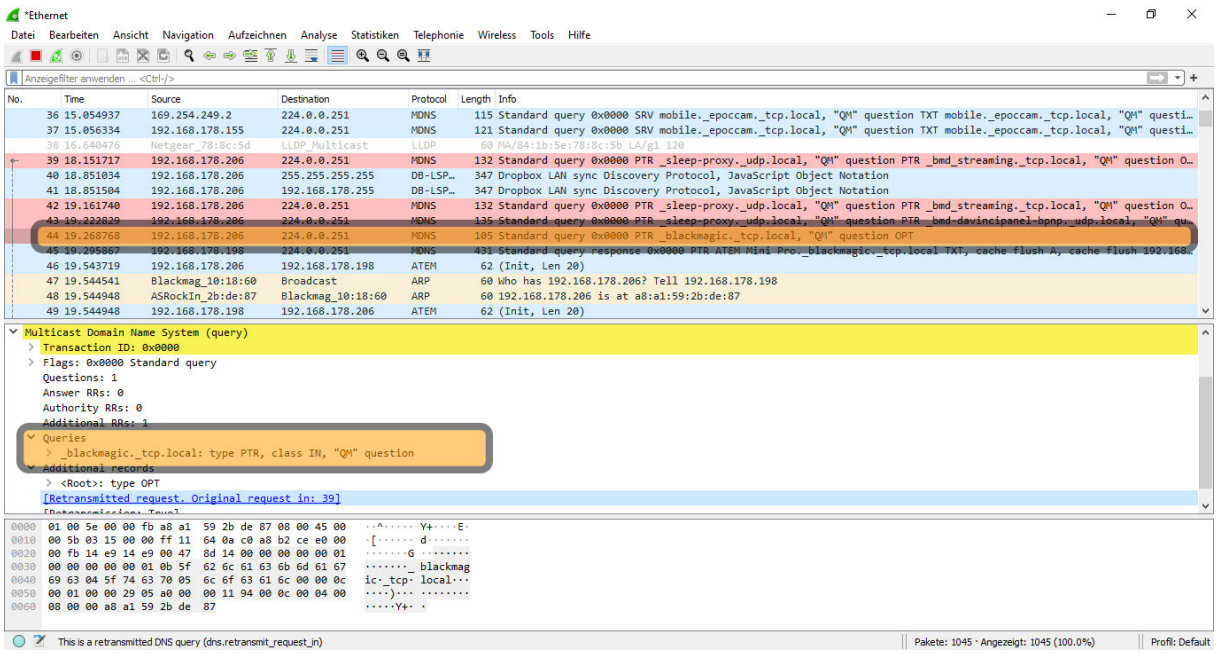
Code 4.7: Gekürzte Loop-Funktion der Controller-Software in `main.cpp`

In die Software des Controllers wird die *ATEMpro* Bibliothek eingebunden. Innerhalb dieser wird die *ATEMbase* und *ATEMstd* eingebunden. So verfüge ich über Funktionen der ursprünglichen *Arduino ATEM Library* von *SKAARHOJ*, sowie meinen Erweiterungen. Der Programmcode selbst lässt sich in zwei Teile trennen, die Kommunikation mit dem Bildmischer und das Abfragen der Steuerelemente. Beide Teile zusammen ergeben den vollwertigen Controller.

### 4.2.1. Kommunikation mit dem Bildmischer

Ein wünschenswertes Feature ist die selbstständige Erkennung von Bildmischern im lokalen Netzwerk. Die *ATEM Software Control* verfügt über eine solche Funktion. Durch Analyse des Netzwerkverkehrs beim Starten der Software wurde schnell deutlich, dass es sich um den *Multicast Domain Name Service (MDNS)* handelt. Das Prinzip dieser Technologie ist, per Multicast im Netzwerk nach IP-Adresse zu fragen. Eingesetzt wird sie häufig in kleinen Netzen, in denen es

keinen DNS-Server gibt. Ein prominente Implementierung des Dienstes ist *Apple Bonjour* [BSI mDNS].




Quelle: Eigene Darstellung

Abbildung 4.1.: Wireshark zeigt empfangenes mDNS Paket

```
void ATEMfound(const char* type, MDNSServiceProtocol proto,
                  const char* name, const byte ipAddr[4],
                  unsigned short port,
                  const char* txtContent) {
  if(name == NULL) {
    Serial.println("Anfrage liefert NULL.");
    return;
  }

  foundAtems++;
  unsigned int index = foundAtems - 1;

  foundAtemNames[index] = name;
  foundAtemAddresses[index][0] = ipAddr[0];
  foundAtemAddresses[index][1] = ipAddr[1];
  foundAtemAddresses[index][2] = ipAddr[2];
  foundAtemAddresses[index][3] = ipAddr[3];
  Serial.println(foundAtemNames[index]);

  found = true;
}
```

Code 4.8: Callback-Routine beim Finden eines ATEM Mischers im Netzwerk

Wird eine *Service Discovery* Anfrage versendet, antwortet ein betreffendes Gerät im Netzwerk mit seiner IP-Adresse und seinem Hostname. In meinem Fall wird im Netzwerk die Anfrage nach einem `_blackmagic` Dienst gesendet und ein verfügbarer ATEM Mini Pro antwortet mit seiner IP-Adresse (siehe Abbildung 4.1). Nach Recherchen konnte ich eine Bibliothek für den Arduino

finden, welche mDNS Service Discovery Pakete versenden kann und diese in den Programmcode implementieren. Wird der Controller eingeschaltet, werden verfügbare Mischer gesucht und dann in einer Liste auf einem Display zur Auswahl gestellt (siehe Abbildung 4.8 in Kapitel 4.3.4).

Sobald das Discovering gestartet ist (siehe Code 4.9), wird beim Auffinden eines Dienstes eine sogenannte Callback-Funktion oder -Routine aufgerufen (siehe Code 4.8), welche zuvor in der Setup-Funktion dem mDNS Dienst mitgeteilt wurde. Diese Funktion liest die IP-Adresse und den Namen des gefundenen Mischers aus und schreibt beides in Arrays. Diese werden in der Loop-Funktion nach dem Start des Controllers ausgelesen und auf einem Display dargestellt.

Wenn auf dem Display dann mithilfe eines Drehencoders ein Bildmischer ausgewählt wird, verbindet sich der Controller mit diesem und der Hauptteil des Programmcodes kann aktiv werden. Dies ist mit einer *State*-Variable gelöst, welche, je nach Zustand des Controllers, wechselt und einen anderen Teil der Loop-Funktion aufruft.

```
if (!EthernetBonjour.isDiscoveringService()) {
      if(!found) {
        if(EthernetBonjour.startDiscoveringService(serviceName, MDNSServiceTCP, 5000)) {
          Serial.println("Discovering started!");
        }
        else {
          Serial.println("Discovering NOT started!");
        }
      }
    }

    EthernetBonjour.run();
```

Code 4.9: Starten der mDNS Service Discovery

### 4.2.2. Abfragen der Steuerelemente

Dieser Teil der Controller-Software kann auch als Hauptteil bezeichnet werden. Hier werden die Steuerelemente ausgelesen (z.B. der Stand eines Faders oder eines Drehreglers) und dieser Wert bei Bedarf dem Bildmischer mitgeteilt. Realisiert wird das Abfragen über eine Funktion, die den Wert eines Faders beispielsweise ausliest und mit einem vorherigen Stand vergleicht. Unterscheidet sich der neue Wert von dem vorherigen um einen bestimmten Wert (hier `5`), gilt der Fader als bewegt und sein neuer Wert wird dem Bildmischer mitgeteilt. Der Wert `5` wurde durch Versuche ermittelt, da das analoge Signal beim Aufbau auf dem Protoboard oft und stark fluktuierte.

```
float getLocalFaderValue(int *faders, float *faderValueHistory, int faderNumber) {
    float faderValueNew = analogRead(faders[faderNumber]);
    float faderValueOld = faderValueHistory[faderNumber];

    if(abs(faderValueNew - faderValueOld) > 5) {
        float val = faderValueNew / 1024;
        faderValueHistory[faderNumber] = faderValueNew;
        return val;
    }
```

```
10      return false;
11  }
```

Code 4.10: Auslesen des Fader-Wertes in der `helpers.cpp`

Die so erhaltenen Werte der einzelnen Komponenten (Fader, Potis und Buttons) werden dann genutzt um die Lautstärke eines Kanals im Fairlight Audiomixer zu beeinflussen. Zunächst wurde dies für einzelne Kanäle und Quellen direkt gemacht um die Funktion zu testen. Zu debug-Zwecken wurde außerdem eine LED an einem digitalen Port des Arduinos bei Änderung eines Wertes kurzzeitig eingeschaltet. In der `for`-Schleife werden alle verfügbaren Fader geprüft und die Werte mithilfe der zuvor beschriebenen Funktion ausgelesen. Liegt eine Änderung vor, wird dieser Wert mit der Funktion `setFaderGain()` an den Mischer gesendet.
Das gleiche Verfahren findet bei den Potentiometern für die Eingangsverstärkung und den Buttons für den live-Status statt.

```
1  for(unsigned int i = 0; i < sizeof(faders)/sizeof(faders[0]); i++) {
2    if(float val = getLocalFaderValue(faders, faderValueHistory, i)) {
3
4      // do something with the faders value
5      AtemSwitcher.setFaderGain(1301, -256, val);
6
7      #ifdef DEBUG
8        Serial.println(val);
9        digitalWrite(debugLed, HIGH);
10     #endif
11   }
12 }
```

Code 4.11: Ändern der Lautstärke eines Fairlight Kanals (debug)

Relativ schnell wurde bei den Tests deutlich, dass das Bewegen des Faders nicht „gleichmäßig“ in der Software Control gespiegelt wurde, sondern eher sprunghaft von Wert zu Wert. Eine Erklärung dafür war die Performance des Arduinos. Sobald viele Aufgaben in der Loop-Funktion durchlaufen werden, nimmt die allgemeine Performance ab. Jedoch konnte diese drastisch gesteigert werden, indem serielle Ausgaben, welche zu Testzwecken an vielen Stellen eingefügt waren, unterbunden wurden. Durch diese Änderung konnten deutlich mehr Pakete für die Änderung eines Faders gesendet werden und der Verlauf der Lautstärke war gleichmäßig.

## 4.3. Hardwarekomponenten und Layout

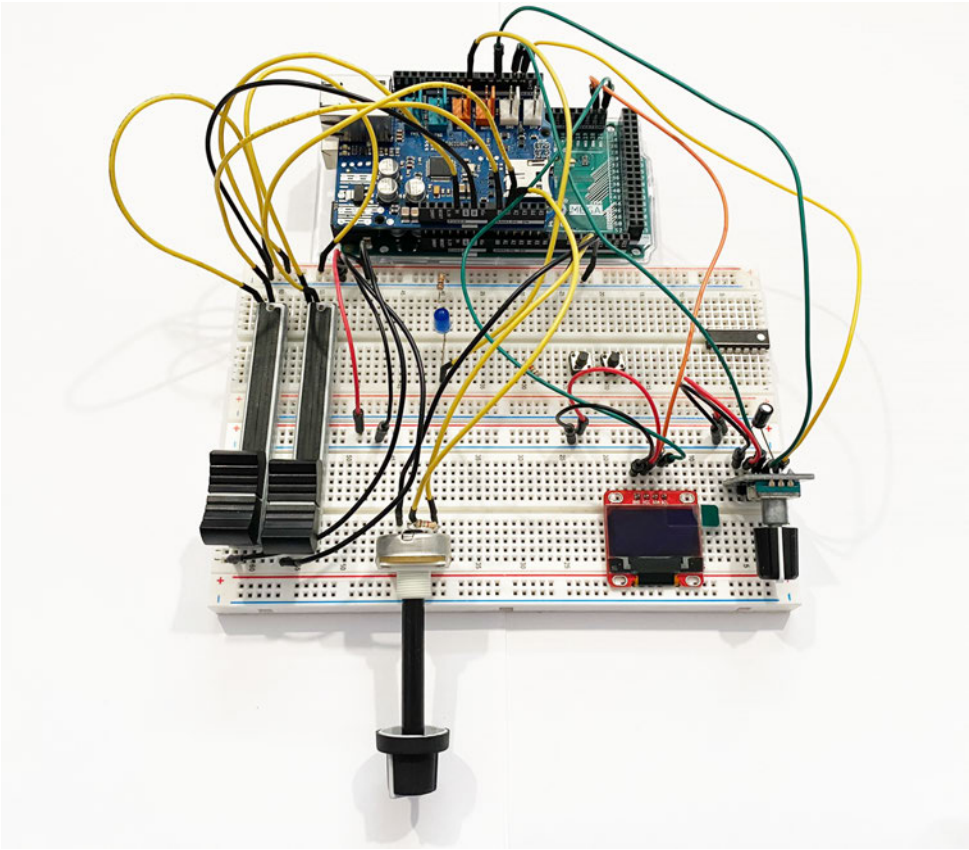
Im folgenden Kapitel gehe ich auf die verwendeten Hardwarekomponenten sowie das Layout und Design des Controllers ein. Als Hardwarekomponenten zählen hierbei die passiven Bauteile wie Schieberegler und Potentiometer, aber auch aktive Komponenten wie Leuchtdioden und Displays. Der verwendete Mikrocontroller wird auch in diesem Kapitel beleuchtet. Die Auswahl der Hardwarekomponenten wurde in direkter Abstimmung mit den benötigten Funktionen getroffen.

Der Aufbau des Audiocontrollers erfolgte zunächst in Teilen auf einem *Protoboard*. Diese auch als Steckbrett oder Breadboard bekannte Platine ermöglicht Testaufbauten und Prototypen ohne

Lötverbindungen zu erstellen. Die vorhandenen Löcher auf der Platine sind reihenweise durchverbunden und ermöglichen einfache Aufbauten mit mehreren Bauteilen ohne Kabelverbindung. Durch das Verbinden zweier Protoboards hatte ich ausreichend Platz zur Verfügung, um große Komponenten, wie die Fader oder das Display, mit dem Arduino zu verbinden und die Funktion zu testen.

In diesem Stand der Entwicklung wurde auch ein Großteil der Software geschrieben und getestet.

### 4.3.1. Arduino Microcontroller



Quelle: Eigene Darstellung

Abbildung 4.2.: Arduino mit Ethernet Shield verbunden mit Protoboard und einigen Bauteilen

Der verwendete Mikrocontroller ist der *Arduino Mega 2560 R3*, welcher auf dem *ATMEGA2560* Mikroprozessor basiert. Die Arduino Entwicklungsboards haben den großen Vorteil, dass sie über alle unterstützenden Features verfügen, die ein sofortiges Entwickeln ermöglichen. Eine USB-Schnittstelle kann zum direkten Programmieren genutzt werden, mehrere Header für analoge und digitale I/Os[6] lassen einen Testaufbau mit Jumper-Kabeln zu (siehe Abbildung 4.2). Insgesamt verfügt der Arduino über 16 Analoge Eingänge, und 54 digitale Ein-/Ausgänge [Arduino Mega]. Die Anzahl der analogen Eingänge war auch der begrenzende Faktor für die Anzahl der

[6] Header (oder Stiftleisten) sind Anschlüsse auf einer Platine, die das gleichzeitige Verbinden von mehreren Kontakten ermöglichen

Fader und Potentiometer, da diese über einen analogen Eingang angeschlossen werden müssen. Es gäbe die Möglichkeit über einen Multiplexer analoge Eingänge zu vervielfachen, was bei Bedarf in nachfolgenden Arbeiten untersucht werden kann.

Die Stromversorgung des Arduinos übernimmt zunächst der USB-Anschluss, welcher auch für das Programmieren verwendet wird. Das wird bei Verwendung von mehreren LEDs jedoch nicht ausreichen, da der Arduino über eine Absicherung des USB-Ports verfügt, welche bei einem Maximalstrom von 500 mA abschaltet. Auch liefert ein üblicher USB-Anschluss nicht ausreichend Leistung. Für den Betrieb ist daher ein Netzteil vorgesehen, was über den DC-Port des Arduinos anschlossen wird und eine stabile Spannungsversorgung von 9 V liefert.

Ein weiterer Vorteil der Header auf dem Arduino ist die Möglichkeit Erweiterungsboards aufzustecken. Ich nutze das *Arduino Ethernet Shield 2* als Erweiterung, um eine Netzwerkverbindung über Ethernet herzustellen. Alle benötigten Verbindungen werden über die Header hergestellt, so z.B. die Spannungsversorgung und die Datenverbindung über die *SPI-Schnittstelle* (Serial Peripheral Interface).

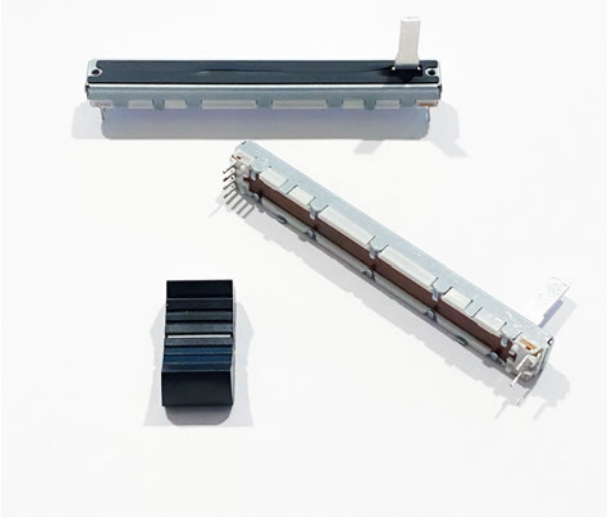
### 4.3.2. Fader

Die Kernfunktionalität, das Steuern der Lautstärke eines Kanals, setzt eine intuitive Bedienung über einen Schieberegler voraus. Daher habe ich für jeden Kanalzug[7] ein *Schiebepotentiometer* (auch Fader genannt) eingebaut (siehe Abbildung 4.3). Dieses passive Bauteil ändert seinen Widerstand über einen definierten Schiebeweg analog zu einem Potentiometer, das den Widerstand über den Drehweg ändert. Für den Einsatz im Controller habe ich einen für Mischpulte typischen Widerstandswert von 10 kΩ gewählt. Die Länge des Schiebeweges ist ein Kompromiss aus Bedienkomfort (was einen längeren Laufweg erfordert) und Platzersparnis (welche für einen kürzeren Laufweg spricht). Des Weiteren haben die Fader eine logarithmische Kennlinie. Der Vorteil davon ist, dass sich das Regeln von logarithmischen Werten, wie der Lautstärke des Kanals welche in dB angegeben ist, für den Bedienenden natürlicher anfühlt. Experimente im Vorfeld mit logarithmischen sowie linearen Widerstandskennlinien der Fader haben das bestätigt.

Da eine Voraussetzung ist acht Audiokanäle gleichzeitig zu steuern, werden 8 Fader an jeweils einen analogen Eingang des Arduinos angeschlossen. Die Fader verfügen über zwei getrennte Kohleschichten innerhalb des Bauteils, was sie „stereotauglich“[8] macht. Für meine Zwecke benötigte ich jedoch nur einen einfachen Widerstand, sodass nur eine Kohleschicht angeschlossen wurde. Der untere Anschluss des Widerstands (im Folgenden Massekontakt oder Masseanschluss genannt) wird mit der Masse des Arduinos verbunden, der obere Anschluss mit dem 5 V Pin des Arduinos. Einer der Analogeingänge wird nun mit dem Schleiferkontakt am Fader verbunden. So liegt je nach Position des Schiebers am Eingang eine Spannung zwischen 0 V und 5 V, welche vom Analog-Digital-Wandler des Mikrocontrollers in Werte zwischen 0 und 1023 übersetzt wird. Dies entspricht einer Genauigkeit von 10 bit.

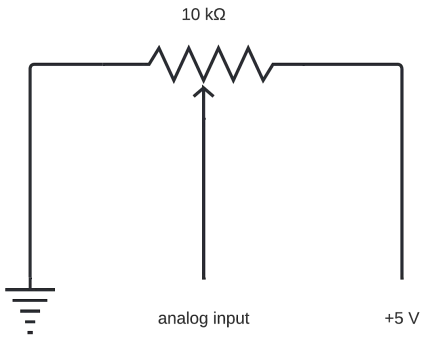
[7] Bei einem Mischpult beschreibt ein Kanalzug die Bedien- und Steuerelemente für einen Audiokanal (z.B. Gain-Regler, Equalizer, Fader, Balance-Regler, etc.)

[8] In analogen Mischpulten werden über zweikanalige Fader Stereosignale geregelt



Quelle: Eigene Darstellung

Abbildung 4.3.: Fader und Faderkappe

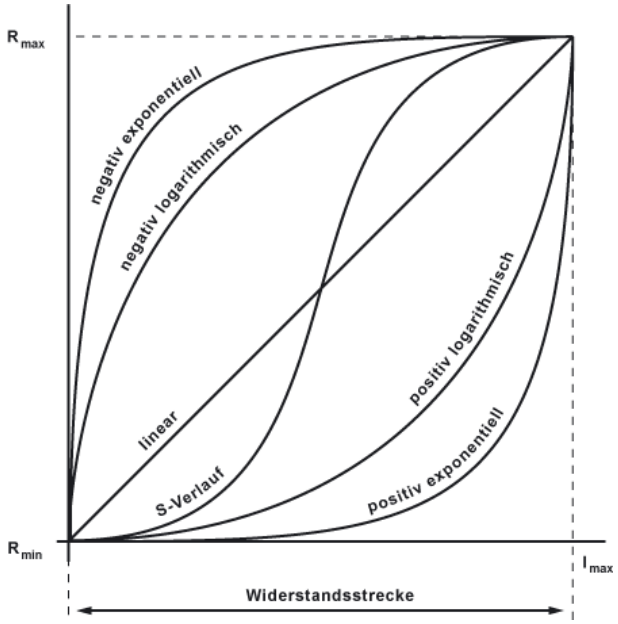



Quelle: Eigene Darstellung

Abbildung 4.4.: Anschlussdiagramm der Potentiometer

### 4.3.3. Drehregler

Als Drehregler verwende ich übliche Potentiometer (oft auch kurz Potis genannt). Die Funktionsweise ist analog zu der des Faders, der Widerstand ändert sich beim Drehen. Die Potis werden als Gain Regler eingesetzt und steuern so die Eingangsverstärkung eines Kanals. Wie beim Fader auch ist dieser Wert in dB angegeben, sodass sich eine logarithmische Kennlinie des Widerstands anbietet.
Durch Experimentieren mit dem Gain-Regler stellte sich heraus, dass sich ein logarithmisches Poti jedoch „falsch anfühlt“, der Verlauf schien verschoben. Wenn das Poti zu einem Dreiviertel gedreht war, wurde lediglich ein Viertel oder weniger des Gains aufgedreht. Der restliche Drehweg des Potis steuerte dann einen sehr großen Gain-Bereich, sodass die Präzision deutlich abnahm. Die Lösung dafür ist eine negativ logarithmische Kennlinie (siehe dazu Abbildung 4.5). So verteilt sich der Gain-Bereich gleichmäßig und für den Bedienenden deutlich „natürlicher“ über den Drehweg des Potis.



Quelle: elektronik-kompendium.de, letzter Zugriff: 15.07.2022 https://www.elektronik-kompendium.de/sites/bau/1011211.htm

Abbildung 4.5.: Verschiedene Kennlinien von Potentiometern

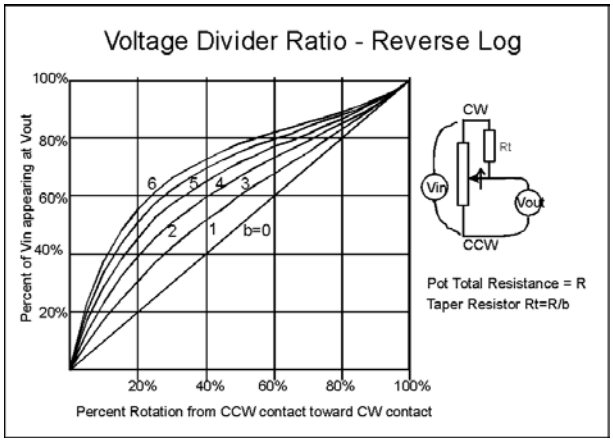
Leider sind Potentiometer mit negativ logarithmischer Kennlinie schwer erhältlich und überdurchschnittlich teurer als Potis mit einer linearen oder positiv logarithmischen Kennlinie. Hierbei hilft jedoch ein elektrotechnischer Trick. Wird ein Widerstand über dem Schleifkontakt und dem 5 V Kontakt angebracht, kann man die lineare Kennlinie eines Potis in eine negativ logarithmische ändern. Zur Berechnung des benötigten Widerstands nutzt man die Formel 4.1 [The Secret Life of Pots].

$$R_t = \frac{R}{b} \tag{4.1}$$

$R_t$ steht hierbei für den einzubringenden Widerstand, $R$ für den Widerstand des Potentiometers und $b$ für den Verlauf der logarithmischen Kennlinie. Die Bezeichnungen wurden aus der Abbildung 4.6 übernommen.

In meinem Fall bot sich die Kennlinie $b = 6$ aus der Abbildung 4.6 an. Der Widerstand wurde dann mit der Formel 4.1 auf 1666 Ω berechnet. Der nächste passende Widerstand aus meinem Bestand war ein 1.5 kΩ Widerstand mit 5 % Toleranz, welcher zu einem zufriedenstellenden Ergebnis führte.

Neben den Potentiometern ist auch ein Drehencoder (oder auch Endlospoti) verbaut. Der Drehencoder funktioniert nach einem anderen Prinzip als ein Potentiometer. Er verfügt über keine Widerstandsschicht über die ein Schleifkontakt fährt, sondern über zwei Tast-Kontakte, welche beim Drehen des Encoders geschlossen werden. Durch einen leichten Versatz in der Positionierung der Kontakte, kann über Abfragen des Zustands der Kontakte die Drehrichtung und die Anzahl der Drehimpulse bestimmt werden. Dieser Drehregler eignet sich gut für die Navigation durch Bildschirmmenüs, wo Eintrag für Eintrag durchlaufen wird oder für die schrittweise Änderung von Werten, wie z.B. des Audio-Delays pro Kanal in Frames. Des Weiteren verfügt der von mir




Quelle: [The Secret Life of Pots], letzter Zugriff: 15.07.2022 http://www.geofex.com/article_folders/potsecrets/potscret.htm

Abbildung 4.6.: Erzeugen einer negativ logarithmischen Kennlinie

verwendete Drehencoder über einen Mitteltaster, der sich durch Drücken des Encoder-Schaftes aktivieren lässt. So können markierte Menüeinträge ausgewählt oder Einstellungen bestätigt werden. Da der Drehencoder für die zwei Tast-Kontakte jeweils einen digitalen Eingang des Arduinos benötigt sowie einen weiteren Kontakt für den Mitteltaster, ist der Einsatz von einem Encoder pro Kanal nicht effizient, da der Arduino nicht über ausreichend digitale Eingänge verfügt. Auch wird für die Abfrage der Werte ein *Interrupt*[9] genutzt, welcher vom Arduino nur auf 6 Eingängen unterstützt wird. Daher habe ich mich entschieden, lediglich einen Drehencoder zu verbauen und diesen für ein Menü und für die Änderung von Werten bei selektierten Kanälen zu nutzen.

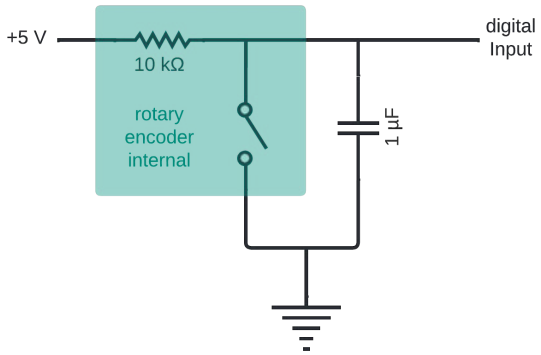
In der Testphase stellte ich fest, dass Drehimpulse des Encoders nicht zuverlässig erkannt werden. In der Theorie sollte ein Drehschritt des Encoders einen Interrupt auslösen und eine Variable um eins erhöhen. Jedoch führte ein Schritt zu einer Erhöhung um deutlich mehr als eins. Es wurden also mehrere Interrupts bei einem Schritt ausgelöst, was die Menüführung unmöglich macht. Das Problem hierzu ist jedoch kein unbekanntes. Ein Kontakt schließt in der Regel nicht „sauber“ und lässt z.B. einen Wert von 0 auf 1 springen, es entsteht hierbei oft das sogenannte *Prellen*. Das Signal springt zwischen den Werten, der Kontakt wird also kurzzeitig wiederholt geschlossen (bzw. geöffnet). Eine einfache und effektive Lösung ist das *Entprellen* mit Hilfe eines Kondensators. Die eingebrachte Kapazität wirkt in Verbindung mit einem *pull-up-Widerstand*[10] im Drehencoder als *Hochpassfilter* (siehe dazu Abbildung 4.7). So werden schnelle Sprünge im Signal nicht vom Eingang des Mikrocontrollers registriert.

### 4.3.4. Optisches Feedback

Um Feedback über live geschaltete Kanäle zu bekommen, werden LEDs an jedem Kanalzug verbaut. Ursprünglich wollte ich diese LEDs in Taster integrieren, sodass man ein direktes Feedback über das Drücken des Tasters erhält. Dies erwies sich aufgrund der Verfügbarkeit kompatibler

[9] Ein Interrupt unterbricht die (sich wiederholende) Routine des Mikrocontrollers, um eine Aktion oder Handlung vorzuziehen, z.B. Drücken eines Tasters leert Speicher

[10] Der Encoder verfügt über einen Widerstand, der den Ausgang der Tast-Schalter im Encoder im nicht-geschlossenen Zustand mit 5V Verbindet, der Eingang des Mikrocontrollers wird so auf „high“ gesetzt, daher *pull-up*

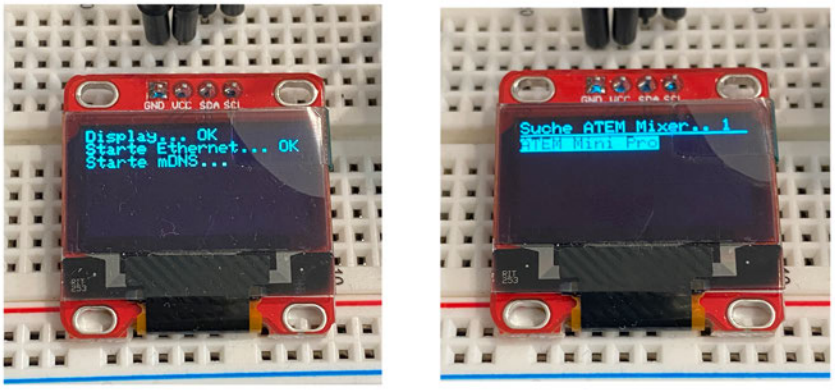



Quelle: Eigene Darstellung

Abbildung 4.7.: Anschlussdiagramm des entprellten Drehencoders

Taster jedoch als schwierig. Mehr dazu im Kapitel 4.3.5. Die LEDs teilen sich eine Masseleitung, welche sie auf Kathodenseite mit der Masse des Arduinos verbindet. Die Anoden wurden jeweils über einen Widerstand von 220 Ω mit einem digital Ausgang des Arduinos verbunden. So können die LEDs einzeln angesteuert werden und den Status eines Kanals signalisieren (live / nicht live).

Ein weiteres optisches Feedback stellt das Display dar. Bereits in vorherigen Projekten konnte ich gute Erfahrungen mit *OLED-Displays* machen. Daher fiel auch hier die Wahl auf ein solches Display, welches sich über den integrierten *SSD1306 display driver* mittels $I^2C$*-Bus* einfach und unkompliziert mit dem Arduino verbinden lässt. So kann über zwei Leitungen (*serial data (SDA)*, *serial clock (SCL)*), sowie 5 V und Masse, das Display angesteuert werden. Es verfügt über 128x64 pixel und ist monochrom.




Quelle: Eigene Darstellung

Abbildung 4.8.: OLED auf Protoboard zeigt Status beim booten (links) und im Netzwerk verfügbare Bildmischer (rechts)

Das Display wird verwendet, um beim Starten des Controllers initiale Informationen auszugeben.

Darüber hinaus, wird ein Auswahlmenü für im Netzwerk verfügbare Bildmischer angezeigt, sowie Statusinformationen bei aktiver Verbindung mit einem Bildmischer (siehe Abbildung 4.8). Auch weitere Werte können über das Display in einem Menü geändert werden, wie z.B. das Audio-Delay.

### 4.3.5. Taster, Buttons

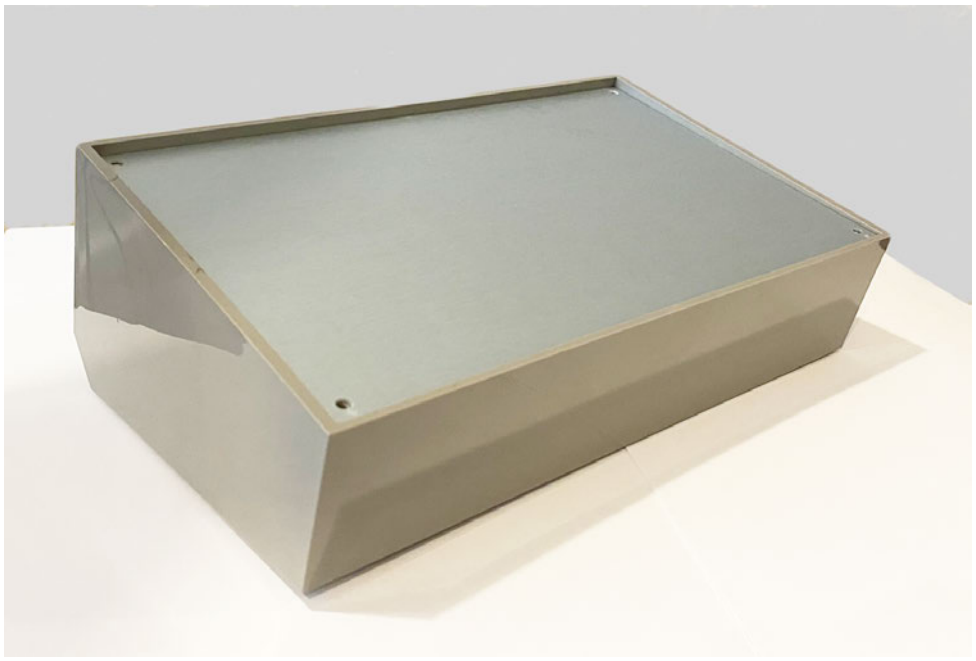
In den ersten Überlegungen zum Design des Controllers, wollte ich ich mich an den Tastern der mittelgroßen Bildmischer-Modelle, wie dem *ATEM Television Studio 4K* orientieren. Da ich bei mehreren Produktionen mit diesem Modell gearbeitet habe, konnte ich die Haptik und das Design kennenlernen. Die Taster verfügen über eine beleuchtete und beschriftete Oberfläche und sind mit einer transparenten Kappe versehen. Ihr Druckpunkt ist sehr fest und es besteht nicht die Gefahr eines versehentlichen Drückens (wie bei Tastern, welche sich sehr leicht drücken lassen). Jedoch sind diese, vermutlich speziell angefertigten Taster, nicht direkt erhältlich und auch durch ihre Größe nicht direkt geeignet für ein portables kleinformatiges Gerät.

Die Wahl ist daher auf Tastatur-Schalter gefallen, welche sich auch in mechanischen Tastaturen finden lassen. Der Druckpunkt ist ähnlich fest und der Formfaktor sehr gering. Jedoch müssen die Taster durch ihre Bauform auf einer Platine befestigt werden und können nicht an der Gehäuseoberfläche angebracht werden, wie z.B. ein Poti, welches man mit einer Kontermutter befestigt.Leider lässt sich die dazugehörige Tastenkappe nur schlecht mit einer LED von unten durchleuchten, da das Material zu dick ist und das Licht nicht in vollem Umfang durchscheint.

### 4.3.6. Gehäuse

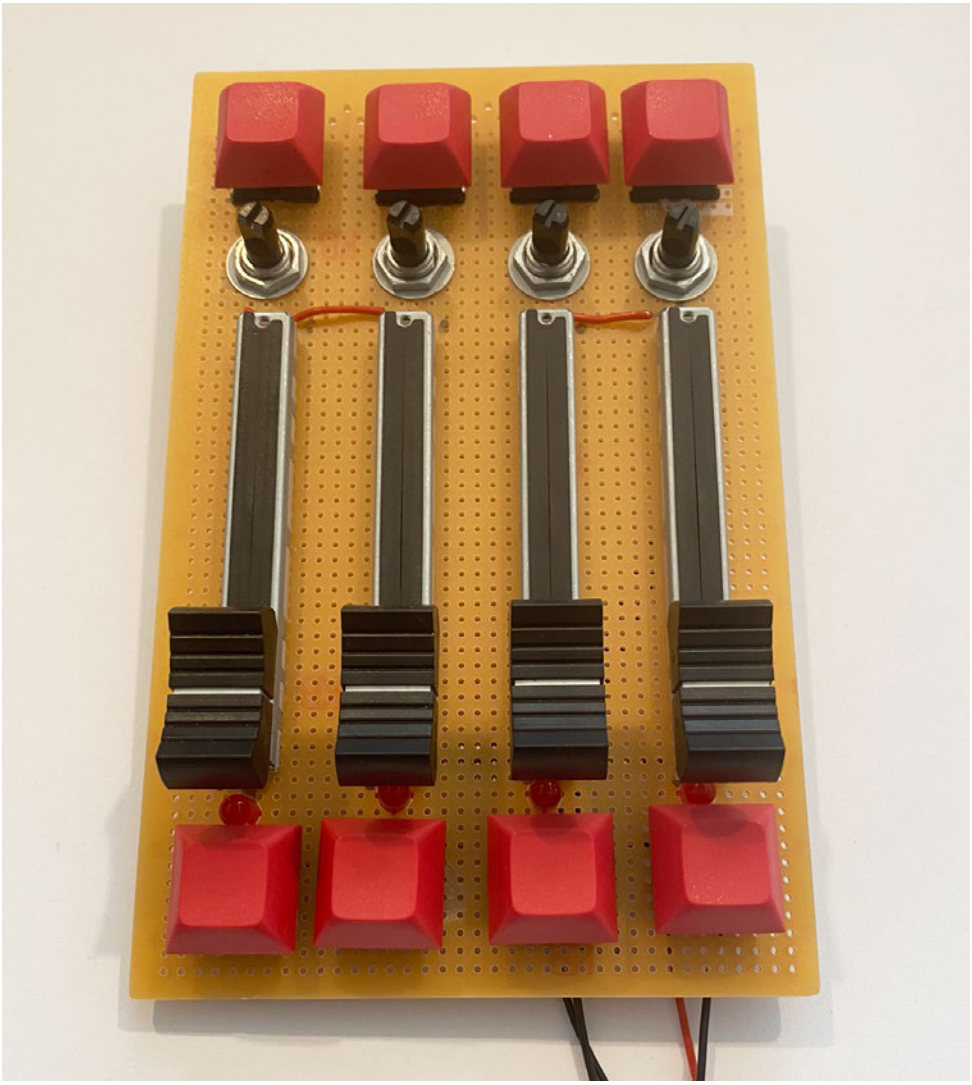
Der ursprüngliche Gedanke, den Controller in einer ähnlichen Größe wie den *ATEM Mini Pro* zu bauen, musste leider verworfen werden. Durch die verschiedenen Bauteilgrößen und den Headern am Arduino, habe ich ein Gehäuse gewählt, welches ausreichend Platz für alle Bauteile bietet. Auch die Entwicklungszeit spielte hierbei eine Rolle.

Der Prototyp (Abbildung 4.10) des Controllers konnte im (Zeit)Rahmen der Bachelorarbeit jedoch nicht in das finale Gehäuse eingebaut werden. Die Fader benötigen für den Einbau einen Schlitz in der Frontplatte des Gehäuses, sodass der eigentliche Fader unterhalb der Platte montiert wird und von oben über einen mit einer Faderkappe besetzten Schaft bedient wird. Das Fräsen des Schlitzes in achtfacher Ausfertigung sowie das Bohren der weiteren Löcher und Aussparungen für die Bauteile konnte nicht im im Zeitrahmen fertiggestellt werden. Die Kompetenz um diese Arbeiten vorzunehmen kann in einer nachfolgenden Arbeit erworben werden oder an einen Dienstleister ausgelagert werden, welcher sich auf das Design von Frontplatten spezialisiert hat.



Quelle: Eigene Darstellung

Abbildung 4.9.: Das Gehäuse des Controllers



Quelle: Eigene Darstellung

Abbildung 4.10.: Prototyp für vier Kanäle, auf separaten Platinen befinden sich die restlichen Komponenten

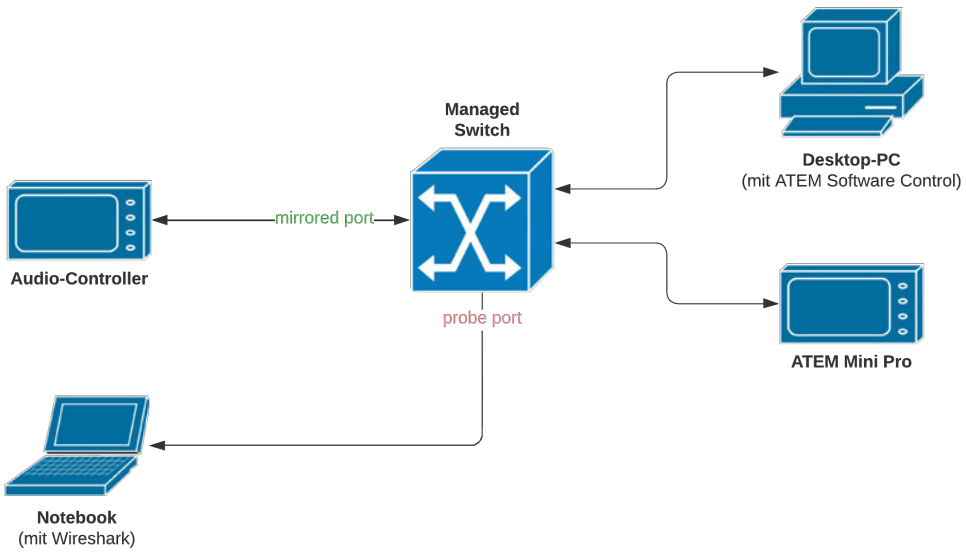
# 5. Funktionstests

Im Rahmen von Tests wurde der Controller auf seine korrekte Funktion und Zuverlässigkeit getestet. Dazu kamen auch die Mittel zum Einsatz, welche in Kapitel 3 bereits zur Analyse des Protokolls genutzt wurden.

Ursprünglich war ein Test bei einem Livestream geplant. Dieser Test musste aufgrund von Zeitmangel entfallen. Im Verlauf der Arbeit an der Bachelorarbeit ergab sich keine Gelegenheit den Audio-Controller bei einem Event zu testen. Für eine Fortsetzung dieser Arbeit wäre dies ein guter Einstiegspunkt.

## 5.1. Verifikation durch Paketanalyse

Während die Paketanalyse mit Wireshark bereits in den Untersuchungen zum Protokoll Anwendung gefunden hat, hilft sie auch bei den Funktionstests des Controllers selbst. Durch Untersuchungen der gesendeten Pakete, kann verglichen werden, ob die Daten dem Protokoll-Standard entsprechen oder ob es Verfälschungen gibt. Dazu wurde ein ähnliches Testsetup aufgebaut.

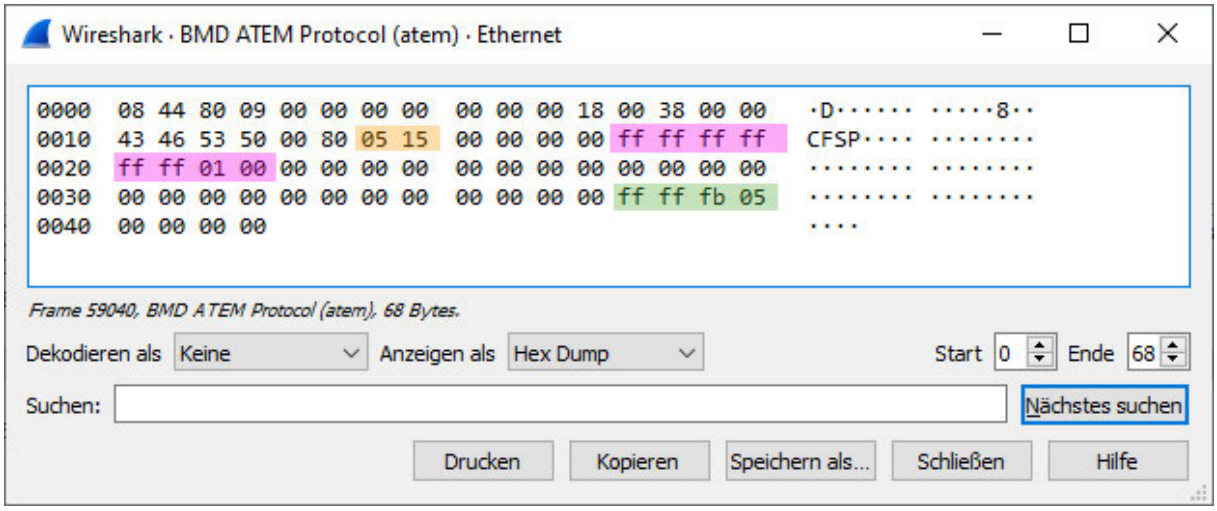



Quelle: Eigene Darstellung

Abbildung 5.1.: Netzwerkdiagramm für die Paketanalyse des Controllers

Der Controller ist in diesem Fall an den Mirror-Port des Switches angeschlossen, sodass ein- und ausgehender Netzwerkverkehr an den Probe-Port und damit an das Notebook gespiegelt wird. Dazu muss der Mirror-Port in den `TX/RX` Modus versetzt werden. So können Pakete vom Controller gesendet werden und vom Mischer beantwortet werden. Allerdings werden Pakete, welche von der *ATEM Software Control* gesendet werden, nicht an Wireshark gespiegelt, da diese Pakete direkt zwischen PC und Bildmischer ausgetauscht werden. So kann es keine Verwechselung zwischen den Paketquellen geben.

Erste Untersuchungen zeigen, dass die Pakete zuverlässig gesendet werden. Sowohl Fairlight Input als auch Source werden korrekt berechnet und ins Paket geschrieben. Auch das betreffende Feld mit dazugehöriger changesFlag wird korrekt übermittelt. In Abbildung 5.2 sieht man orange hinterlegt `0515` was in dezimaler Darstellung `1301` entspricht, dem Namen des Fairlight Inputs von Mikrofoneingang 1 (vergleiche dazu Tabelle 3.1). In violett hinterlegt wird die Fairlight Source übertragen, in diesem Fall die (dez) `-65280`, was einer Stereoquelle entspricht. Der grün hinterlegte HEX-Wert `FFFFFB05` ist eine Zweierkomplement-Darstellung von `-1275`, was der Bildmischer zu −12.75 dB parst.




Quelle: Eigene Darstellung

Abbildung 5.2.: Screenshot von Wireshark mit markierter HEX-Darstellung einzelner Felder

Die Gegenprobe mit der *ATEM Software Control* zeigt jetzt, dass der Fader vom Kanal Mikrofon 1 auf −12.75 dB steht. Der Befehl wurde also korrekt vom Bildmischer erkannt (siehe Abbildung 5.3).

Nach diesem Verfahren wurden alle implementierten Befehle getestet und auf korrekte Funktion überprüft. Bei den beiden vom Bildmischer aus gesendeten Befehlen `FASP` und `FMTl` wurde zusätzlich im Arduino überprüft, ob die Werte korrekt geparsed werden. Dazu bieten sich zwei Möglichkeiten an. Die Ausgabe über die serielle Schnittstelle, also mittels des `Serial.print()`-Befehls und die Kontrolle mithilfe der Bauteile wie LED oder Display. Letztere ist auch eine finale Lösung für die Audio-Tallys. Im Falle des live Schalten eines Kanals, zeigt eine LED dieses an.


Quelle: Eigene Darstellung

Abbildung 5.3.: ATEM Software Control zeigt Fader und Kanal-Lautstärke nach Test

## 5.2. Ergebnisse

Die Tests ergeben eine fehlerfreie Umsetzung der Befehle in der Bibliothek. Alle untersuchten Pakete erwiesen sich als fehlerfrei und wurden vom Empfänger korrekt interpretiert. Die Funktion der Bibliothek ist damit gegeben. Bei den Versuchen mit den Fadern hat sich allerdings gezeigt, dass die verwendeten Fader unpräzise sind. Durch den kurzen Laufweg wird eine verhältnismäßig große Werteänderung in kleinem Bereich durchgeführt. Dies hat einen Mangel an Präzision zur Folge. Auch spielt hierbei die Auflösung von `10 Bit` des analog/digital-Wandlers eine Rolle. Mit höherer Auflösung steigt die Präzision der analogen Bauteile.

Bis zur Fertigstellung der Bachelorarbeit konnten nicht alle Funktionen bis ins Detail getestet werden. Durch die begrenzte Zeit mussten sich die Tests auf die wesentlichen Funktionen, welche in Kapitel 2.2 dargestellt wurden, beschränken. Eine ausgiebige längerfristige Testung, auch bei Einsatz während einer Livestream-Produktion, wird noch im Nachgang an die Bachelorarbeit erfolgen.

Zusammenfassend lässt sich sagen, dass der Controller im Rahmen seiner Möglichkeit gut funktioniert und stabil Pakete sendet und empfängt. Auch über einen längeren Zeitraum von mehreren Stunden bleibt die Verbindung zum Bildmischer aufrecht und lässt sich weiter nutzen.

# 6. Zusammenfassung und Ausblick

Ziel der Bachelorarbeit ist es, einen Audio-Controller zur Steuerung von *Blackmagic Design ATEM* Bildmischern, bzw. zur Steuerung der Audio Engine innerhalb dieser. In diesem Kapitel wird der entstandene Controller evaluiert und ein Fazit gezogen.

In der vorliegenden Bachelorarbeit wurde ein funktionsfähiger Audio-Controller entwickelt. Sein Design wurde in Verbindung mit den Anforderungen entwickelt und mit Hardwarekomponenten umgesetzt. Ein Arduino-Board stellt dabei die Grundlage zur Verfügung. Die benötigte Software wurde von mir programmiert und im Laufe der Entwicklung angepasst und verbessert. Der finale Controller ist in seinem Prototypen-Stand durch Untersuchungen auf Funktion getestet worden, die Ergebnisse wurden analysiert.

Ich konnte zeigen, dass es noch immer möglich ist, die *SKAARHOJ Arduino ATEM library* zu nutzen, die aktuelle Bildmischer-Firmware wird weiterhin unterstützt. Eine Erweiterung der Bibliothek war möglich und brachte die Funktion von neuen Befehlen des ATEM Protokolls in die Bibliothek ein. Die so entstandene neue *ATEMpro library* wird auf open-source Basis über die Plattform *GitHub* anderen Nutzer:innen zur Verfügung gestellt. Dieses Vorgehen ist in meinem Augen selbstverständlich, da ich in dieser Arbeit von open-source Software anderer Autoren Gebrauch machen konnte. Ein „Zurückgeben“ der gewonnenen Erkenntnisse an die entwickelnde Community ist wichtig und hält das Konstrukt open-source Software aufrecht.

Die Entwicklung eines intuitiven Hardware-Designs erfolgte aus dem Mangel an intuitiver Bedienbarkeit des *ATEM Mini Pro*. Die Funktionen, welche dort nicht oder nur unzureichend vorhanden sind, wurden auf dem Audio-Controller umgesetzt. Dabei konnten nicht alle Funktionen, welche den Bildmischer sinnvoll ergänzen, umgesetzt werden. Beschränkungen durch die Hardware und die Zeit zur Entwicklung ließen mich die Funktionen durch Priorisierung auf die wichtigsten und hilfreichsten kürzen.

Ein finaler Controller wurde fertiggestellt, er ist jedoch in einem Prototypen-Stand einzuordnen. Weitere Entwicklungszeit und Tests sind notwendig, um ihn auf den Stand eines gut einsetzbaren Geräts zu bringen. Im Zuge der Arbeit an dem Controller stellten sich einige weitere Funktionen als fehlend heraus. So müsste, um das Projekt fortzuführen, über eine größere Hardware-Plattform nachgedacht werden. Weitere Drehregler bzw. analoge Bauteile müssten ausgelesen werden. Dazu müssten weitere analoge Eingänge auf dem Mikrocontroller vorhanden sein. Weiterhin sollten einige wichtige Software-Änderungen vorgenommen werden. Das ATEM Protokoll verfügt über Befehle, welche beim Verbinden mit einem Bildmischer dem Client die aktuelle Konfiguration mitteilen. Diese sollte vom Controller erfasst und dynamisch darauf reagiert werden können. Ein Beispiel dafür ist die Kanalkonfiguration als Mono- oder Stereoquelle. In diesem Zusammenhang wären auch motorisierte Fader denkbar, damit eine Änderung durch ATEM

Software Control auf dem Controller widergespiegelt wird. Auch könnte sich der Fader so beim Verbinden auf die aktuelle Lautstärke einstellen.

Abschließend lässt sich sagen, dass der Audio-Controller für ATEM Bildmischer eine sinnvolle und hilfreiche Erweiterung für kleine Bildmischer wie den *ATEM Mini Pro* ist. Trotz des engen Rahmens der Bachelorarbeit, konnte ein funktionsfähiger Prototyp entwickelt werden. Dieser bildet eine sehr gute Basis für eine weitere Ausarbeitung.

# Literaturverzeichnis


[SKAARHOJ Protocol] Skaarhoj.com, *BlackMagic ATEM Switcher Protocol*, letzter Zugriff: 12.06.2022
https://www.skaarhoj.com/discover/blackmagic-atem-switcher-protocol

[Rotary Encoder] Nikolaus, Stefan, 2016, *Rotary Encoder*, letzter Zugriff: 12.06.2022
https://www.nikolaus-lueneburg.de/2016/02/rotary-encoder/

[GNU GPL v3] Free Software Foundation, Inc., 2007, *GNU GENERAL PUBLIC LICENSE*. letzter Zugriff 29.06.2022
https://www.gnu.org/licenses/gpl-3.0.html

[UDP] Dipl.-Ing. (FH) Stefan Luber / Dipl.-Ing. (FH) Andreas Donner, 2019, *Was ist UDP (User Datagram Protocol)?*, letzter Zugriff: 30.06.2022
https://www.ip-insider.de/was-ist-udp-user-datagram-protocol-a-789006

[OSI Modell] Elektronik-Kompendium.de, *OSI-Schichtenmodell*, letzter Zugriff: 30.06.2022
https://www.elektronik-kompendium.de/sites/kom/0301201.htm

[Open Switcher] Martijn Braam, 2021, *The Open Switcher Documentation*, letzter Zugriff: 11.07.2022
https://docs.openswitcher.org/

[The Secret Life of Pots] R.G. Keen, 1999, *The Secret Life of Pots*, letzter Zugriff: 08.07.2022
http://www.geofex.com/article_folders/potsecrets/potscret.htm

[Arduino Mega] *Arduino Mega 2560 Rev3 — Arduino Official Store*, letzter Zugriff: 10.07.2022
https://store.arduino.cc/products/arduino-mega-2560-rev3

[LibAtem] GitHub - LibAtem, *LibAtem*, letzter Zugriff: 11.07.2022
https://github.com/LibAtem/LibAtem

[NRK Sofie] GitHub - nrkno, *NRK Sofie TV automation*, letzter Zugriff: 11.07.2022
https://github.com/nrkno/Sofie-TV-automation

[Sofie ATEM connection] GitHub - nrkno, *Sofie: The Modern TV News Studio Automation System (ATEM connection library)*, letzter Zugriff: 11.07.2022
https://github.com/nrkno/sofie-atem-connection

[Blackmagic Unternehmen] Blackmagic Design - blackmagicdesign.com *Unternehmen* lezter Zugriff: 12.07.2022
https://www.blackmagicdesign.com/de/company

[Interview Patty] Dominic Powell, 2017, *Blackmagic Design founder Grant Petty on how a "burning revelation" led him to create his $300 million video technology company*, letzter Zugriff: 15.07.2022
https://www.smartcompany.com.au/entrepreneurs/influencers-profiles/blackmagic-design-founder-grant-petty-on-how-a-burning-revelation-led-him-to-create-his-300-million-video-technology-company/

[BMD Forum] *What does „ATEM" mean?* letzter Zugriff: 12.07.2022
https://forum.blackmagicdesign.com/viewtopic.php?f=4&t=111608

[Arduino Forum] forum.arduino.cc, 2017, *How do I use C++ to code the Arduino boards?*, letzter Zugriff: 13.07.2022
https://forum.arduino.cc/t/how-do-i-use-c-to-code-the-arduino-boards/493409/9

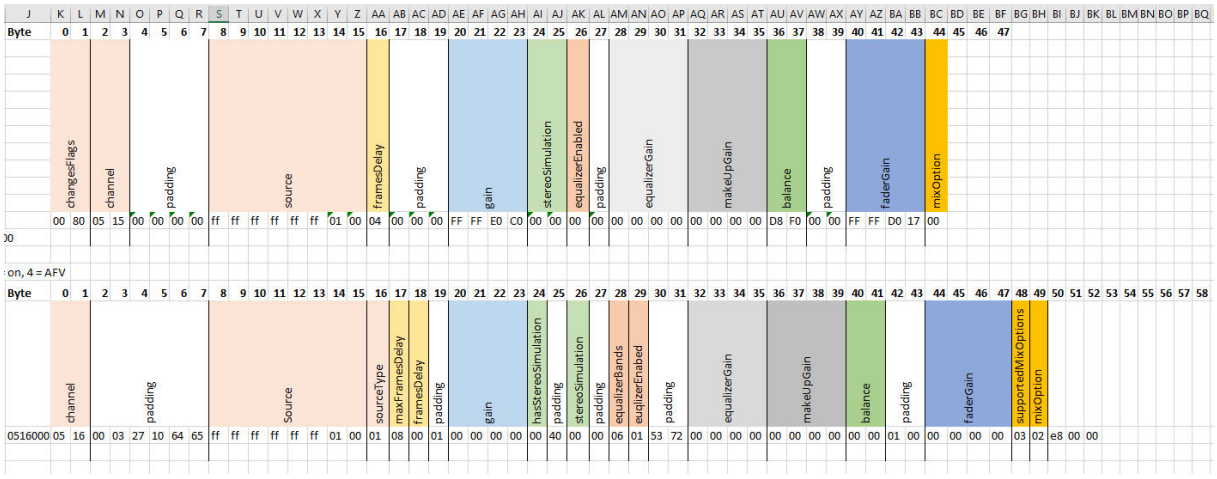
[BSI mDNS] www.bsi.bund.de, *CERT-Bund Reports: Offene mDNS-Dienste*, letzter Zugriff: 13.07.2022
https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Cyber-Sicherheitslage/Reaktion/CERT-Bund/CERT-Bund-Reports/HowTo/Offene-mDNS-Dienste/Offene-mDNS-Dienste_node.html

[BMD Media Archive] blackmagicdesign.com - Blackmagic Media Archive, 2019, *Blackmagic Design kündigt neuen ATEM Mini an* letzter Zugriff: 13.07.2022
https://www.blackmagicdesign.com/de/media/release/20190913-01

[Futurebiz] Jan Firsching, 2020, *7,5 Mrd. Stunden: Live-Streaming auf Twitch, YouTube & Facebook Gaming wachsen um 92 %*
https://www.futurebiz.de/artikel/live-streaming-wachstum-92-prozent

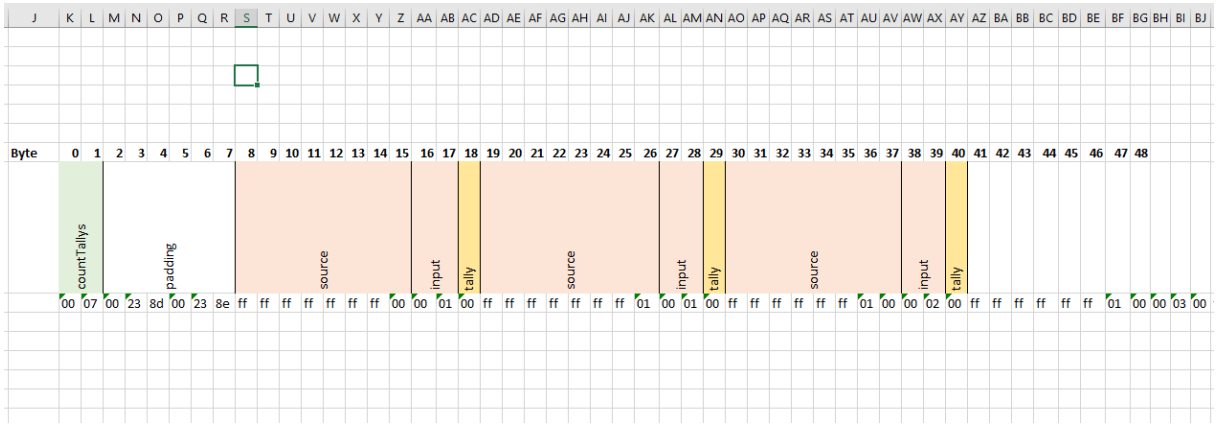
# A. Anhang

## A.1. Excel Tabellen zur Analyse




Quelle: Eigene Darstellung

Abbildung A.1.: Manuelles Parsing in Excel für den Befehl CFSP und FASP




Quelle: Eigene Darstellung

Abbildung A.2.: Übersicht und Parsing des FMTl Befehls in Excel

## Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorthesis mit dem Titel

**Entwicklung eines Audio-Controllers für ATEM Bildmischer**

selbstständig ohne fremde Hilfe gefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quelle kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum Eike Börgeling